# Formale Spezifikation reaktiver Systeme mit einer Sicherheitsfachsprache

Von der Fakultät für Mathematik, Naturwissenschaften und Informatik
der Brandenburgischen Technischen Universität Cottbus

zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation

vorgelegt von

Diplom-Ingenieur

Thomas Mertke

geboren am 01.06.1969 in W.-Pieck-Stadt Guben (jetzt Guben)

Gutachter: Prof. Dr.-Ing. Monika Heiner

Gutachter: Prof. Dr.-Ing. Ulrich Berger

Gutachter: Prof. Dr.-Ing. Hans-Michael Hanisch

Tag der mündlichen Prüfung: 24.Oktober 2003

Vorwort

Die Grundlagen zu dieser Arbeit entstanden während meiner Tätigkeit am Lehrstuhl Automatisierungstechnik der Brandenburgischen Technischen Universität Cottbus. Ein besonderer Dank gilt deshalb dem damaligen Lehrstuhlinhaber Herrn Prof. Dr.-Ing. H. Meier sowie Frau Prof. Dr.-Ing. M. Heiner für die Initiierung des zugrunde liegenden Forschungsprojekts und die wissenschaftliche Betreuung der Arbeit. Herrn Prof. Dr.-Ing. U. Berger, dem jetzigen Lehrstuhlinhaber, danke ich für die Übernahme der Betreuung gerade in der letzten Phase der Arbeit.

Ein weiterer Dank gilt den Herren T. Menzel und P. Deussen, die gerade im Bereich der Informationstechnik und der Temporalen Logik viele notwendige und nützliche Hinweise geben konnten. Die Zusammenarbeit zwischen der klassischen Automatisierungstechnik und der Informatik hat hier zu vielen neuen und wertvollen Erkenntnissen geführt. Herrn L. Scharf danke ich speziell für die Programmierung des Editors der Sicherheitsfachsprache und des Compilers, die trotz der vielen Änderungen und Erweiterungen einen hervorragenden Reifegrad erreicht haben.

Den Mitarbeitern des Lehrstuhls Automatisierungstechnik Herrn R. Schulze, Herrn K. Kreusch, Herrn U. Steffen und Herrn K. Henning danke ich für die zahlreichen Gespräche und Hinweise zum Gelingen dieser Arbeit.

Nicht zuletzt gilt ein besonderer Dank meiner Familie, allen Freunden und Kollegen, die mich bei der Durchführung dieser Arbeit unterstützt und motiviert haben. Viele mussten die Höhen und Tiefen in der Zeit dieser Arbeit miterleben. Ich möchte ihnen für ihr Verständnis und ihre Geduld danken.

Ingolstadt / Cottbus, Januar 2003 Thomas Mertke

## Bemerkungen zum Schriftbild

Einige Passagen dieser Arbeit beinhalten formale Definitionen der Sicherheitsfachsprache, sowie Beispiele und Erläuterungen zu ihrer Anwendung. Um diese Darstellungen vom eigentlichen Text optisch abzugrenzen, werden diese Bereiche besonders dargestellt.

Normaler Fließtext, Arial, 12pt

*Verbale Anforderungen, die mit dem SFS-Editor erstellt wurden,*
*Times New Roman, 12pt, kursiv, eingerückt, mit Rahmen*

Metatext der Sicherheitsfachsprache
Arial, 12pt, kursiv, eingerückt, mit Rahmen

```
Temporallogische Formeln, die durch den SFS-Editor bzw.
den Compiler erzeugt wurden
Courier New, 12pt, eingerückt, mit Rahmen
```

```
Formale Sprachdefinitionen in Backus-Naur-Form
Courier New, 12pt
```

Terminalsymbole innerhalb einer Sprachdefinition
Times New Roman, 12pt,

## Kurzfassung

Rechnersysteme und automatische Steuerungen durchdringen in immer stärkerem Maße unser tägliches Leben. Sie helfen uns bei der Lösung komplizierter, zeitaufwändiger und monotoner Aufgaben, sie geben uns die Möglichkeit, den Menschen von körperlich schweren und gefährlichen Arbeiten zu entlasten. Rechnersysteme dringen heute auch immer mehr in Bereiche vor, für die es vor wenigen Jahren noch als undenkbar galt, die Kontrolle an eine Maschine zu übergeben. Grundvoraussetzung für einen solchen Migrationsprozess ist neben der Entwicklung immer kleinerer und leistungsfähigerer Hardware die Entwicklung der zugehörigen Software, ohne die der eingesetzte Rechner oder das Steuerungssystem nutzlos bleiben würde.

Ein methodisches und wohl überlegtes Vorgehen bei der Erstellung von Software, das so genannte Software-Engineering, war bisher eine Domäne der Informatik. Jedoch auch in der Automatisierungstechnik erfordern die zunehmende Komplexität, das steigende Sicherheitsbedürfnis der Endanwender sowie die stärkere Verbreitung „konventioneller“ Rechentechnik (inklusive der eingesetzten Programmiersprachen) neue Herangehensweisen bei der Entwicklung, Realisierung und Qualitätssicherung von Steuerungssoftware. Ein wesentlicher Problempunkt hierbei ist die vollständige und korrekte Beschreibung der zu lösenden Steuerungsaufgabe. Heute existieren eine Vielzahl unterschiedlicher (teilweise auch genormter) Darstellungsformen, jedoch gibt es kein einheitliches und vor allem kein durchgängiges Werkzeug für die Beschreibung der gewünschten Steuerungsfunktionen. Je nach Anwendungsgebiet, Anwenderkenntnissen und Problemstellung sind die verschiedenen Darstellungsformen so speziell, dass sie nur für Fachleute zu handhaben sind. Der Entwurf einer solchen Sprache muss sich an den Bedürfnissen der Anwender orientieren.

Mit der in dieser Arbeit vorgestellten Sicherheitsfachsprache (SFS) ist eine allgemeinverständliche und eindeutige Formulierung von Steuerungsaufgaben möglich. Durch ihre Nähe zur natürlichen Sprache wird sie ihren grundlegenden Zielstellungen gerecht: eine fachübergreifende Kommunikation zwischen den verschiedenen Beteiligten eines Projektes sowie eine ebenenübergreifende Kommunikation in den verschiedenen Ebenen einer Steuerungsstruktur. Als drittes herausragendes Merkmal bietet die Sicherheitsfachsprache die Möglichkeit, unerwünschte und verbotene Situationen des Steuerungssystems zu beschreiben. Somit ist ein weites Anwendungsfeld in vielen Problembereichen gegeben. Trotz allem ist die Sicherheitsfachsprache formal genug, um eine eindeutige Problemdarstellung zu gewährleisten.

Innerhalb dieser Arbeit wird die Sicherheitsfachsprache wie folgt dargestellt:

Der Abschnitt 1 beinhaltet grundlegende Aussagen zur Korrektheit und Sicherheit von Software und stellt Wege und Mittel dar, wie sich die Qualität von Software erhöhen lässt.

Im Abschnitt 2 werden typische Anforderungen an ein formales Beschreibungsmittel zusammengefasst, das den Gegebenheiten eines Einsatzes in der modernen Steuerungstechnik gerecht werden soll.

Aus diesen Anforderungen wird im Abschnitt 3 die Sicherheitsfachsprache abgeleitet. Dazu werden die steuerungstechnischen Anforderungen kategorisiert, die Parameter der SFS werden abgeleitet und erläutert. Außerdem wird in diesem Abschnitt die

Überführung der Anforderungen in die formale Basis der SFS, die Temporale Logik, dargestellt.

Im Abschnitt 4 schließt sich eine Darstellung der prototypischen Realisierung der Sicherheitsfachsprache und deren Anwendung an. Hier wird erläutert, wie die konkreten steuerungstechnischen Objekte (Sensoren, Aktoren) in die SFS eingebunden werden, wie die Anforderungen erstellt und in die temporallogischen Formeln überführt werden. Außerdem wird ein Verfahren vorgestellt, mit dem ein möglichst umfassendes Set an Anforderungen erstellt werden kann.

Der Abschnitt 5 enthält eine Fallstudie, mit deren Hilfe die umfassenden Möglichkeiten der SFS an einem realen Beispiel demonstriert und nachgewiesen werden sollen. Wegen des großen Umfangs dieser Fallstudie werden in diesem Abschnitt jedoch nur ausgewählte Problembereiche präsentiert, weitere Details können dem Anhang entnommen werden.

Abschließend werden im Abschnitt 6 die wichtigsten Ergebnisse der Arbeit zusammengefasst. Einige Aussagen zu weiteren Entwicklungsmöglichkeiten und Anwendungsfeldern der Sicherheitsfachsprache runden die Ausführungen ab.

**Abstract**

Computer systems and automatic controls have an increasing influence upon our daily life. They are useful for the solution of complex, time-consuming and monotonous problems. They give us the possibility to relieve employees of physically serious and dangerous work. Today we can find computers in some areas, where it was still unthinkable a few years ago, to transfer control to a machine. The main condition for this migration process is, apart from the development of smaller but more efficient hardware, the development of the accompanying software. Without this software the computer or the control system would remain useless.

A methodical and well-considered procedure for the development of software, the so-called software engineering, is a domain of computer science by now. Though the increasing complexity in automation engineering, the rising needs of safety of the user as well as the growing spread of „conventional“ computer hardware (used programming languages included) require new approaches in development, realisation and quality assurance of control software. In this case an essential problem is the complete and correct description of the desired control functions. Today we have a lot of different (even partly standardised) forms of representations, but there is no homogeneous and above all no area-overlapping tool for the description of the desired control functions. Depending on application area, user knowledge and problem definition these forms of representations are that unique, that they can only be used by experts. The development of a description language has to be oriented to the users needs.

With the „Safety-oriented Technical Language” (in German: Sicherheitsfachsprache - SFS), which is presented in this work, the formulation of control problems in an easy, clear and common way of understanding is possible. Because it comes near the natural language it fulfils the main objectives: an overlapping communication between different persons, who are involved in a project and an overlapping communication between different levels of a hierarchical control structure. As a third main feature the SFS offers the possibility to describe unwanted and prohibited situations of a control system. Thus a wide application field to solve different problems is given. On the other side the SFS is well defined enough to ensure a clear problem specification.

Within this work the SFS is represented as follows:

Section 1 contains basic statements concerning the correctness and safety of software and shows ways and means, how to increase the quality of software.

In section 2 typical requirements for a formal description tool are summarized, which should fulfil the conditions of a use in modern control technology.

In section 3 the „Safety-oriented Technical Language” is derived from these requirements. For thus the technical requirements will be classified, the parameters of the SFS are derived and explained. The transfer of requirements into the formal base of the SFS (temporal logic) is additionally represented in this section.

In section 4 a representation of the technical realisation of the SFS its use is given. It is explained, how the concrete control technical objects (sensors, actuators) are

included into the SFS, how the requirements can be explained and transferred into formulas. In addition a method will be explained, how to arrange a complete set of requirements.

Section 5 contains a case study, to demonstrate the extensive possibilities of the SFS using a real example. Because of the high complexity of this case study only selected problems are presented. Further details can be taken from the appendix.

Finally section 6 summarises the most important results. Some statements about further possibilities for development and use cases of the SFS are closing this work.

## Inhalt

## Abbildungen

## Tabellen

## Verzeichnis der verwendeten Symbole und Abkürzungen

| | |
|---|---|
| A | - Anforderung |
| B, $B_1$, $B_2$ | - Bedingung |
| F, $F_1$, $F_2$ | - Folgerung |
| BI | - Beobachtungsintervall |
| S | - Startzustand |
| E | - Endzustand |
| Z | - Zielzustand |
| DE, DEs, DEe | - Anforderungskategorien: demand, simple demand, extended demand |
| PR, PRs | - Anforderungskategorien: prohibition, simple prohibition |
| PO, POe | - Anforderungskategorien: possibility, extended possibility |

| | |
|---|---|
| BDE | - Betriebsdatenerfasssung |
| BNF | - Backus-Naur-Form |
| CNC | - Computerized Numerical Control |
| CPU | - Central Processing Unit |
| CTL | - Computation Tree Logic |
| EBNF | - Extended Backus-Naur-Form |
| FMEA | - Fehler-Möglichkeits- und Einflussanalyse (engl. Failure Mode and Effect Analysis) |
| MDE | - Maschinendatenerfassung |
| MMI / HMI | - Mensch-Maschine-Interface (engl. Human Machine Interface) |
| NC | - Numeric Control |
| RC | - Robot Control |
| SFS | - Sicherheitsfachsprache |
| SPS / PLC | - Speicherprogrammierbare Steuerung (engl. programmable logic controller) |

Verwendete Symbole zur Darstellung der Temporalen Logik

| | CTL-Darstellung | CTL-Darstellung im SFS-Editor (Standard-ASCII-Zeichensatz) |
|---|---|---|
| Konjunktion, logisches ‚Und’ | $\wedge$ | & |
| Disjunktion, logisches ‚Oder’ | $\vee$ | nicht definiert |
| Negation, logisches ‚Nicht’ | $\neg$ | ! |
| Implikation, Konditional, ‚wenn ..., dann ...’ | $\rightarrow$ | -> |
| Äquivalenz, Bikonditional, ‚nur wenn ..., dann ...’ | $\leftrightarrow$ | <-> |
| Öffnende Klammer | ( | ( |
| Schließende Klammer | ) | ) |

Symbole der BNF / EBNF

`<...>` - nichtterminale Symbole

`::=` - Zuweisung

„ ... “ - terminale Symbole

`|` - Alternative

`(* ... *)` - Kommentare

`,` - Trennzeichen für Aufzählungen

`undefined` - undefinierte Zeichenketten

# 1 Einleitung und Ausgangssituation

Bei der heutigen kundenorientierten Fertigung von Maschinen und Anlagen stellt die Erzeugung der zugehörigen Steuerungssoftware einen wesentlichen Zeit- und Kostenfaktor dar. Software ist zu einem integralen Bestandteil zeitgemäßer Infrastrukturen geworden und viele technische Systeme sind ohne die Leistungsfähigkeit moderner Rechentechnik nicht mehr denkbar. Der Einsatz klassischer festverdrahteter Sicherheitssysteme nimmt deutlich zugunsten rechnergestützter Systeme ab [Hala98]/[Habl98]. Andererseits kommt es immer wieder zu Ausfällen solcher Systeme. Die Medien berichten über Unfälle und Katastrophen in sicherheitssensiblen Anwendungen (Stichworte: Flugzeug-Unglücke [Meff99], Ariane-5-Erststart [Habl98] und[1]). Oftmals lässt sich das Fehlverhalten nach gründlicher Analyse auf Fehler in den Steuerungssystemen - und speziell auf Mängel in der verwendeten Software - zurückführen (siehe auch [Leve95]). Die zunehmende Komplexität moderner Software, das steigende Sicherheitsbedürfnis der Endanwender [Hala98] und die diesen Gegebenheiten entgegenstehenden Vorbehalte gegenüber programmierbaren elektronischen Systemen [Meff99] erfordern neue Methoden und Herangehensweisen, mit denen die Korrektheit und somit die Sicherheit von Steuerungsprogrammen verbessert werden kann. Wie stark wir in unserem täglichen Leben von der Zuverlässigkeit von Software abhängig sind, zeigen die Tatsachen, dass eine Softwarefehlerquote von 0,1% bedeuten würde, dass täglich 16.000 Briefe bei der Post verloren gingen, täglich 18 Flugzeuge abstürzten und stündlich 22.000 Schecks falsch gebucht würden [Balz96].

[Hala98] beschreibt, dass heute zur Überprüfung der Korrektheit von Steuerungssoftware zwei Verifizierungsschritte notwendig sind: einerseits der (mathematische) Nachweis der Korrektheit gemäß der gegebenen Spezifikation, andererseits müsste auch gezeigt werden, dass die gegebene Spezifikation auch „die mehr oder weniger vage formulierten Forderungen oder gar unausgesprochenen Gedanken des Spezifizierers erfüllt“. [Litz98] führt ebenso an, dass „... Verfahren benötigt (werden), die die Erkennung von Fehlern oder Unvollständigkeiten in der Spezifikation unterstützen“.

## 1.1 Softwarefehler und ihre Ursachen

### *1.1.1 Begriffsbestimmung*

Stark vereinfacht gesagt, ist ein Programm dann korrekt bzw. fehlerfrei, wenn es sich während seiner gesamten Betriebszeit, also in allen Situationen, denen es ausgesetzt ist, exakt so verhält, wie der Anwender es erwartet. Der Anwender entwickelt bei der Benutzung des Programms eine gewisse Erwartungshaltung, die unter anderem auf einer bestimmten Zielvorgabe, z. B. der Lösung einer gestellten Aufgabe, beruht. Stimmt das aktuelle Verhalten des Programms nicht mit seiner Erwartungshaltung überein, so wird im Allgemeinen von einem Programmfehler gesprochen.

[1] siehe auch www.rvs.uni-bielefeld.de/publications/incidents und www.esrin.esa.it/htdocs.tidc/Press/Press96/ariane5rep.html

Das Fehlverhalten eines Programms lässt sich folgendermaßen grob klassifizieren [Litz98]:

a) das Programm führt eine Aktion aus, die nicht erwartet wurde, diese Fehler werden als aktive Fehler bezeichnet,
b) das Programm führt eine erwartete Aktion nicht aus, diese Fehler werden als passive Fehler bezeichnet.

Bei einem aktiven Fehler werden also Steuerungsfunktionen ausgelöst, ohne dass die programmgemäß festgelegten Bedingungen erfüllt sind. Von einem passiven Fehler spricht man, wenn Steuerungsfunktionen blockiert sind, obwohl alle programmgemäß festgelegten Bedingungen erfüllt sind.

Die Ursachen für ein solches Verhalten können vielgestaltig sein. Zunächst muss man festhalten, dass Software ein anderes Ausfallverhalten als Hardware hat. Typische Probleme der Hardware, wie Abnutzung und Verschleiß treten bei Software (abgesehen von einer „moralischen“ Alterung) prinzipiell nicht auf. Software verändert nicht von selbst ihre Eigenschaften, sie ist inhärent fehleranfällig [Hala98]/[Habl98]. Vielmehr muss man davon ausgehen, dass Software, die sich im laufenden Betrieb als mangelhaft herausstellt, diese Eigenschaften bereits bei der Inbetriebnahme besaß. Softwarefehler sind also ausschließlich systematische Fehler, die aus der Planung, dem Entwurf und der Programmierung hervorgehen.

H. Balzert beschreibt in [Balz96] die Ursachen für solche Programmierfehler: Irrtümer, Schnitzer, (über-) individuelle Denkfallen sowie individuelle Fehler. Im einfachsten Fall entstehen Fehler durch Schreibfehler bei der Codierung des Programms, d. h. es werden Variablen falsch geschrieben, ähnlich lautende Variablennamen werden vertauscht oder es werden unbewusst falsche Programmoperationen niedergeschrieben. Viele dieser *‚syntaktischen’* Probleme lassen sich durch eine geeignete Unterstützung bei der Programmierung aufdecken und vermeiden (z. B. durch geeignete Compilerroutinen oder Inspektion des Codes).

Bestimmte Problemstellungen führen interessanterweise bei vielen Programmierern zu Unsicherheiten und Fehlern, vor allem, wenn es darum geht, mit Situationen umzugehen, mit denen sie nicht genügend vertraut sind (z. B. selten benutzte physikalische Größen oder Einheiten, inverse Problemstellungen, mehrdeutige Beschreibungen).

D. Hablawetz nennt in [Habl98] folgende Ursachen für mangelnde Software-Qualität:

- deutlich gestiegener Aufgaben- und Programmumfang,
- falsche oder unvollständige Spezifikation,
- unzureichender Ausbildungsstand im Software-Engineering,
- zu geringer Informationsaustausch zwischen den an Projekten beteiligten Abteilungen,
- mangelnde oder unvollständige Dokumentation,
- unzureichendes und unsystematisches Testen ohne vorher klar definierte Testfälle.

Die Deutsche Gesellschaft für Qualität bekräftigt in [DGQ92] diese Aufstellung, indem sie als Ursachen für Software-Qualitätsprobleme schlechte Planung, schlechte Kommunikation, unvollständige unklare Anforderungen, sich ändernde Anforderungen, Komplexität, unzureichende Transparenz sowie eine unzureichende Ausbildung anführt.

Diese Aufstellung macht eindrucksvoll deutlich, dass es im Zusammenhang mit der Korrektheit von Software keine einzelnen und losgelösten Ursachen gibt, sondern dass es vielmehr immer eine komplexe Verknüpfung mehrerer Ursachen gibt, die für die Entstehung unzureichender Software verantwortlich sind. Als ein wesentlicher Bereich der Fehlerentstehung lässt sich jedoch eindeutig der Zeitraum im Vorfeld der eigentlichen Softwareerstellung herausarbeiten.

A. Kohring stellt bereits in [Kohr91] und [Kohr93] fest, dass Softwarefehler in frühen Phasen der Entwicklung entstehen, aber erst viel später entdeckt und beseitigt werden können. Als Ursache hierfür wird auch die unzureichende Klärung der Aufgabenstellung genannt. Es sind keine universellen und allgemeinverständlichen Beschreibungsmethodiken verfügbar, die umfassend geeignet sind und akzeptiert werden.

Halang und Konakovski gehen in [Hala98] genauer auf diese Problematik ein. Sie führen aus, dass Softwarefehler bereits beim Entwurf und der Programmierung gemacht werden, also systematischer Natur sind. Die größte Fehlerquelle sind Spezifikationsfehler und nichtberücksichtigte Situationen, wie sie bei einer extrem großen und somit nicht mehr handhabbaren Zahl diskreter Systemzustände auftreten können.

H. Balzert beschreibt in [Balz96] die besondere Rolle des Systemanalytikers im Kontext moderner Softwareerstellung: dieser muss sich aktiv um die Ermittlung der Anforderungen kümmern. Hierfür stehen ihm nur manchmal schriftlich formulierte Anforderungen zur Verfügung, die durch eigene Interviews und Befragungen ergänzt werden müssen. In der Regel werden Produktanforderungen durch den Auftraggeber weder systematisch, strukturiert noch vollständig festgelegt. In den eventuell vorhandenen Dokumenten wird oft zwischen abstrakten Angaben und konkreten Wünschen hin- und hergesprungen. Dies führt dazu, dass die Gesprächspartner lange Zeit kein vollständiges Modell des zu entwickelnden Systems im Kopf haben.

### *1.1.2 Die Softwarekrise in der Steuerungstechnik*

Ein wesentlicher Grund für das fehlerhafte Verhalten von Software lässt sich also bereits in der Phase der Vorbereitung der Softwareerstellung finden: einer mangelhaften Spezifikation der Software. Eine Ursache hierfür liegt wiederum darin, dass in der ingenieurtechnischen Praxis die Anwendung formaler Methoden nicht in dem heute notwendigen Maße erfolgt. Diese Tatsache ist unter anderem historisch begründet: der Ausbildungsstand der involvierten Ingenieure und die für die Softwareerstellung verwendeten Methoden haben nicht mit der rasanten Entwicklung der verwendeten Hardware Schritt gehalten.

Mit der Entwicklung und Anwendung der Relais- und Schütztechnik bestand seit 1938 (vergleiche [Rein96]) zum ersten Mal die Möglichkeit komplexe boolesche

Funktionen für mathematische Berechnungen (1941 – Zuse Z3) und zur Steuerung von Maschinen heranzuziehen. Diese wurden durch die geeignete Verschaltung der Öffner- und Schließer-Kontakte von Relais verwirklicht, und später durch mechanische Zeit- und Zählglieder erweitert. Die „Programmierung“ erfolgte hierbei durch die Verdrahtung, eine Veränderung der Funktion der Steuerung war also nur mit hohem Aufwand möglich. Da der räumliche, technische und auch finanzielle Aufwand dieser elektrischen Steuerungen recht hoch war, wurden auch nur relativ kleine und abgeschlossene Steuerungslösungen verwirklicht. Wegen des hohen Aufwandes bei der Realisierung der gewünschten Funktionalität und wegen deren geringen Komplexität konnte man jedoch im Allgemeinen gut durchdachte und funktionssichere Lösungen erhalten.

Mit der Einführung der kontaktlosen und nahezu verschleißfreien Halbleitertechnik änderte sich das Bild schlagartig. Man war in der Lage, weitaus mehr Funktionen zu einem günstigen Preis zu realisieren. Jedoch ging dies mit einer weitaus schwierigeren Änderbarkeit (bis hin zur Neuerstellung von Leiterkarten) und einem höheren Schulungsaufwand sowie einer komplizierten Fehlersuche einher. Die Programmierung erfolgte weiterhin durch die Verschaltung von Bauelementen, deren Funktionsweise nun jedoch von außen nicht mehr unbedingt nachvollziehbar war.

Als 1968 in den USA die erste kommerzielle speicherprogrammierbare Steuerung (MODICON 084 - Richard E. Morley) auf den Markt gebracht wurde, kam es wiederum zu einer sprungartigen Veränderung der Situation. Zunächst verfügten diese Systeme zwar nur über begrenzte Rechen- und Speicherkapazitäten, jedoch war es nur eine Frage der Zeit, und es begann eine ähnlich rasante Entwicklung wie bei Personalcomputern. Moderne industrielle Steuerungssysteme sind ähnlich leistungsfähig wie Bürorechner, selbst Steuerungssysteme auf reiner PC-Basis werden heute eingesetzt. Diese hohe Leistungsfähigkeit führt jedoch auch zu dem Wunsch, immer mehr und komplexere Funktionalitäten auf diesen Systemen auszuführen. Applikationen, die man vor Jahren noch für undurchführbar hielt, sollen nunmehr durch Steuerungssysteme bewältigt werden. Folglich werden Ingenieure und Techniker heute mit Problemstellungen beauftragt, deren Lösung neben dem ingenieurwissenschaftlichen Wissen ein weitaus höheres Verständnis der Informatik voraussetzt, als dies bisher der Fall war. Steuerungstechniker werden zunehmend als Softwareentwickler tätig. Die dabei verwendeten Methoden und Verfahren haben jedoch offensichtlich mit der rasanten technischen Entwicklung nicht Schritt gehalten.

### *1.1.3 Spezifikation von Steuerungssystemen*

G. Futschek visualisiert in [Futs89] den Softwareentstehungsprozess mit folgender Darstellung (Abbildung 1):

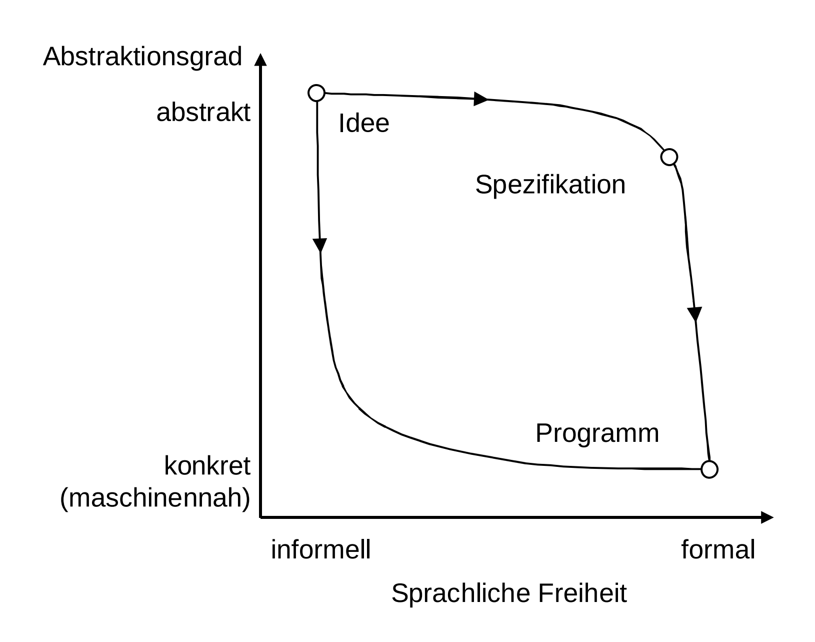



*Abbildung 1 - Entstehung eines Programms aus einer Idee*

Ausgehend von einem Anfangszustand, in dem es eine für den Menschen verständliche, abstrakte und informelle Idee für ein Programm gibt, besteht das Ziel darin, zu einem formalen, auf dem Rechner ablauffähigen Programm zu kommen. Für die Erreichung dieses Ziels gibt es viele Wege, jeder Pfad dorthin ist ein ganz spezieller Programmentwicklungsprozess. Darunter gibt es solche, die ausgehend von der Idee sehr schnell auf eine maschinennahe und dennoch informelle Darstellung überführt werden (z. B. so genannte quick-hack's, Prototypen). Ein anderer Weg, der von ihm und vielen anderen favorisiert wird, besteht darin, möglichst lange in einer abstrakten Ebene zu verbleiben, dort zu einer formalen Darstellung zu gelangen und erst zum Schluss auf eine konkrete Kodierung des Problems zu wechseln. Hierbei definiert die Spezifikation, was das Programm leisten soll. Gleichsam ist durch das Programm der Erfüllungsgrad der Spezifikation bestimmt [Futs89].

Die Entwicklung eines Steuerungsprogramms für eine zu automatisierende Anlage kann man als einen Prozess betrachten, bei dem geforderte und gewünschte Eigenschaften in Software implementiert werden. Diese Soll-Eigenschaften werden üblicherweise in einem Pflichtenheft abgelegt, das durch Kooperation von Auftraggeber und Steuerungstechniker erstellt wird. Ein Pflichtenheft (also die Spezifikation) enthält die notwendige Funktionalität des Programms, die erforderlichen Reaktionen auf festgelegte Ausnahmesituationen, aber auch Festlegungen, welches Programmverhalten unbedingt vermieden werden muss.

Konventionell (also per Hand) erstellte Programme weisen jedoch häufig nicht nur die gewünschten und erforderlichen Eigenschaften auf. Die eingangs genannten Katastrophen und die persönliche Erfahrung jedes Einzelnen zeigen, dass sich Software mitunter anders verhält, als es eigentlich gefordert und erwartet wird. Dieses

„Fehlverhalten“, wie es dann immer genannt wird, beruht unter anderem darauf, dass die erzeugte Software auch zusätzliche „versteckte“ Eigenschaften hat. Diese wurden nicht spezifiziert und vom Programmierer auch nicht direkt implementiert, sie sind dennoch vorhanden. Viele dieser Eigenschaften können harmlos sein, es gibt mit großer Wahrscheinlichkeit jedoch auch solche, die während des Betriebes zu riskanten und gefährlichen Reaktionen des Programms führen können. Ein weiterer typischer Fehler ist, dass das gewünschte Verhalten nicht in allen Kombinationen sichergestellt wird, dass es also Spezialfälle gibt, in denen das Programm versagt.

Der Spezifikationsprozess von Software kann folgendermaßen veranschaulicht werden. In dem betrachteten System, in das die Software integriert werden soll, gibt es üblicherweise eine Anzahl von variablen Größen, die während des Betriebes verschiedene Werte (Zustände) einnehmen können. Durch Kombination dieser Variablenzustände entsteht eine (oftmals sehr große, aber dennoch endliche) Menge von Zuständen, die das gesamte System einnehmen kann. Innerhalb dieser Menge von (theoretisch einnehmbaren) Systemzuständen gibt es nun einerseits solche, die erwünscht, d. h. normale Betriebszustände des Systems sind (Abbildung 2). Auf der anderen Seite gibt es auch Systemzustände, die unbedingt vermieden werden müssen, weil sie eine Gefährdung des Systems oder der Umwelt (z. B. des Bedieners) darstellen. Das zu entwickelnde Steuerungsprogramm soll nun den Zustandsraum des Systems so eingrenzen, dass einerseits die gewünschten Systemzustände (und zwar in einer geforderten Reihenfolge) erreicht und andererseits die unerwünschten Systemzustände nicht mehr eingenommen werden.

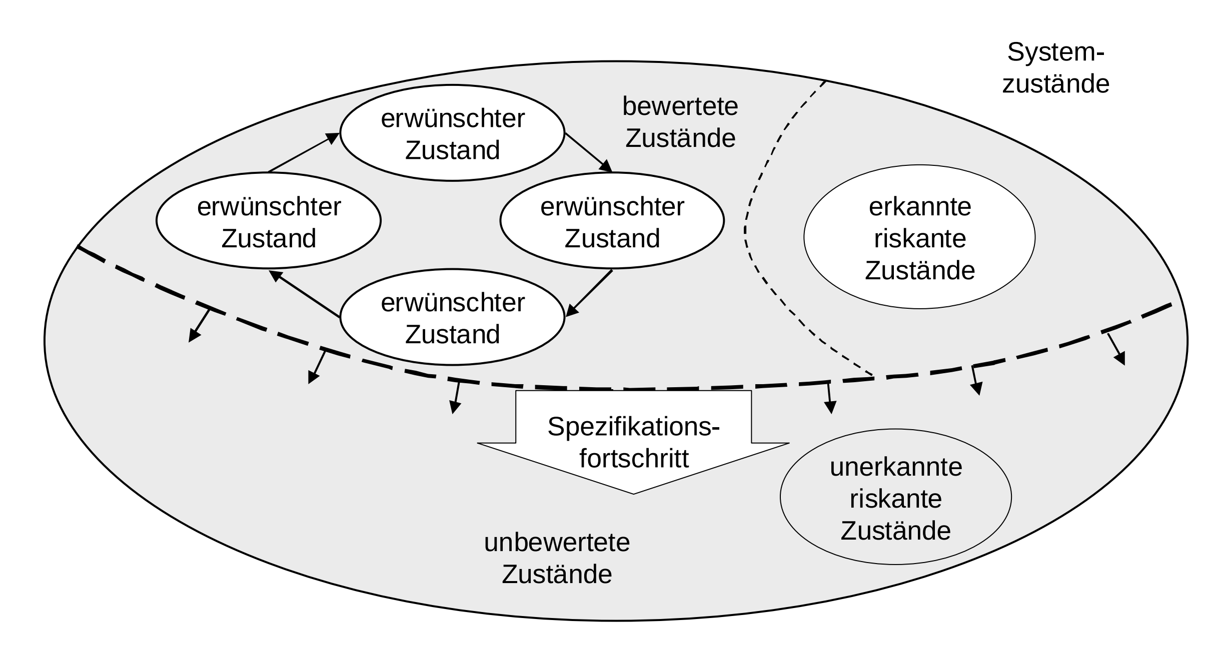



*Abbildung 2 - Spezifikation als Bewertung von Systemzuständen*

Bei der Spezifikation des Programms sollte idealerweise jeder einzelne Systemzustand dahingehend bewertet werden, wie sich das Programm verhalten soll, wenn das System diesen Zustand einnimmt, bzw. ob das System diesen Zustand überhaupt einnehmen darf. Durch diese Bewertung und die Festlegung der Reaktion des Steuerungsprogramms auf diese Systemzustände wird das gewünschte Verhalten des Steuerungsprogramms festgelegt.

Jedoch ist bei Systemen technisch relevanter Größe eine vollständige Spezifikation, also die Betrachtung aller Systemzustände, nicht mehr ohne weiteres möglich. Infolgedessen bleibt eine Menge von Systemzuständen oft unbewertet (unspezifiziert). Hierbei entsteht ein Problem, weil es innerhalb der Menge der unbewerteten Systemzustände auch wiederum riskante Zustände geben kann, die zudem noch unerkannt sind.

Wird die festgeschriebene Spezifikation in reale Software überführt, so weist das Programm oftmals die bereits beschriebenen Fehlfunktionen auf. Die Gründe hierfür sind in zwei Tatsachen zu finden (Abbildung 3):

1. unvollständige Umsetzung des gewünschten Verhaltens, d. h. einige der geforderten Reaktionen werden nicht ausgeführt, nachfolgend als Fehlerklasse a bezeichnet,

2. Implementierung zusätzlicher Eigenschaften, d. h. das Programm ist zu unerwarteten Reaktionen in der Lage, nachfolgend als Fehlerklasse b bezeichnet.

Diese Klassifizierung ist auch konform zu der in Abschnitt 1.1.1 getroffenen Einteilung, die Fehlerklasse a beinhaltet nun die passiven Fehler, Fehlerklasse b beinhaltet die aktiven Fehler.

Die Fehlerklasse b kann man unter Beachtung der Spezifikation noch weiter untergliedern in:

- b1: die unerwartete Ausführung von Aktionen, obwohl diese bei der Spezifikation bereits als unerwünscht oder sogar gefährlich eingestuft wurde,
- b2: die unerwartete Ausführung von Aktionen, die zu dem Zeitpunkt überhaupt noch nicht berücksichtigt und beurteilt wurden.

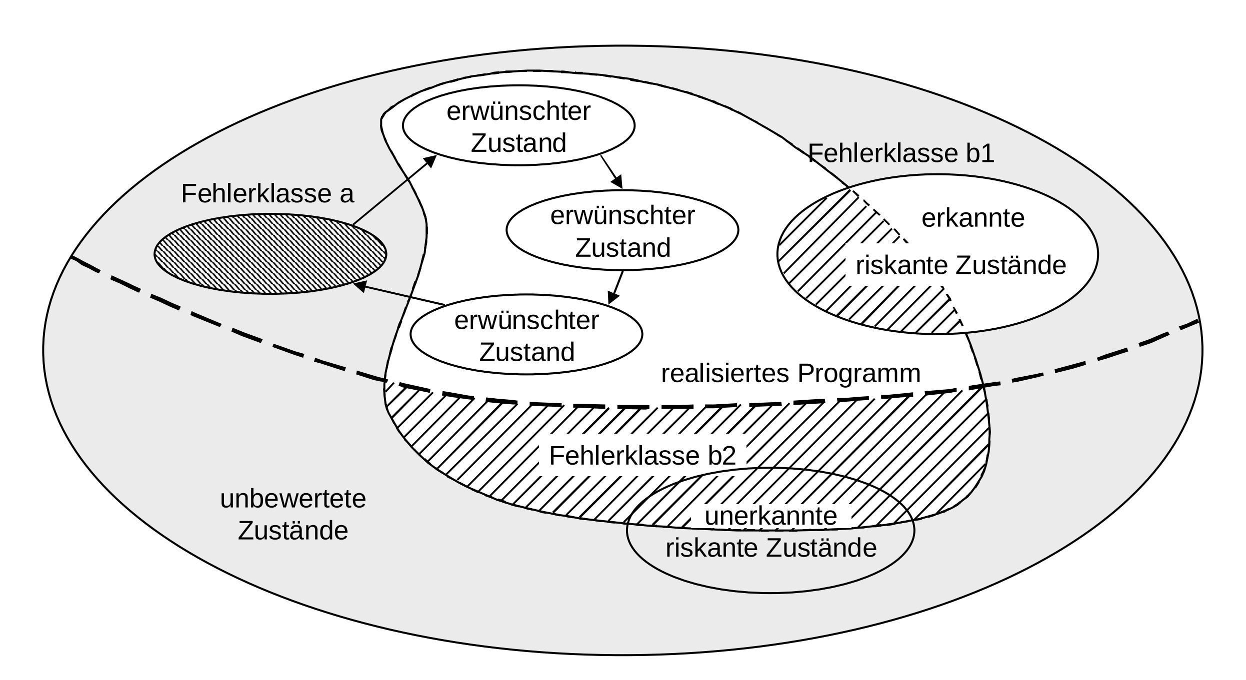



*Abbildung 3 - Ursachen für Softwarefehler*

Somit entstehen bei der Erstellung von Software verschiedene kritische Bereiche, die in der Abbildung 3 schraffiert hervorgehoben wurden. Ziel der anschließenden Inbetriebnahmephase muss es dann sein, diese unbeabsichtigt erzeugten Unzulänglichkeiten zu erkennen und zu beseitigen. Die Überprüfung der Eigenschaften wird heute üblicherweise durch verschiedene Tests durchgeführt. Diese Tests beinhalten hauptsächlich die Inspektion der geforderten Solleigenschaften des Programms, die Kontrolle der Programmreaktionen auf zuvor festgelegte Fehlerszenarien sowie - wenn auch nur stichprobenartig - die Vermeidung gefährlicher Situationen.

Ausgehend von diesen Anforderungen wurde die Sicherheitsfachsprache entwickelt, die drei hauptsächliche Zielstellungen verfolgt:

1. Die eindeutige Kommunikation zwischen den Beteiligten an einer technischen Aufgabenstellung soll ermöglicht werden. Die Mitarbeiter kommen hierbei oftmals aus unterschiedlichen technischen Bereichen und haben deshalb auch einen verschiedenartigen technischen Hintergrund.

2. Die Darstellung der Anforderungen in verschiedenen Stufen des Entwicklungsprozesses soll mit einem einzigen Hilfsmittel durchführbar sein. In den einzelnen Phasen eines Projektes gibt es abweichende Anforderungen an ein Beschreibungsmittel.

3. Das Beschreibungsmittel soll die Darstellung von Anforderungen, die mit den bereits bekannten Mitteln nicht darstellbar sind, ermöglichen. Momentan gebräuchliche Darstellungsformen in der Steuerungstechnik erlauben nur die Darstellung des gewünschten Verhaltens, wogegen die Darstellung unerwünschten Verhaltens nicht möglich ist.

## 1.2 Parameter der Softwarequalität

Ebenso wie für andere technische Systeme ist der Begriff der Qualität auch auf Software und somit auch auf die in dieser Arbeit betrachteten Steuerungsprogramme anwendbar. Qualität wird übereinstimmend definiert als „die Gesamtheit von Merkmalen einer Einheit (bzw. eines Systems) bezüglich ihrer Eignung, festgelegte und vorausgesetzte Erfordernisse zu erfüllen“.

Auch in Bezug auf Software lassen sich Merkmale und Parameter identifizieren, deren Erfüllung bzw. Nichterfüllung Aussagen über die Qualität der betrachteten Software gestattet. Die folgende Übersicht führt einige dieser Parameter näher aus ([Hore89], [DGQ86], [Litz99]).

Korrektheit, Zuverlässigkeit (reliability): Fähigkeit der Software, Aktionen so auszuführen, wie sie in der Spezifikation definiert wurden,

Robustheit: Fähigkeit der Software, sich auch in ungewöhnlichen Situationen vernünftig zu verhalten, dies betrifft vor allem das Verhalten der Software in zuvor nicht spezifizierten und bewerteten Situationen,

Erweiterbarkeit, Anpassbarkeit: Fähigkeit der Software, sich leicht an veränderte Anforderungen oder Spezifikationen anzupassen, d. h. wie leicht sich Änderungen und Erweiterungen implementieren lassen,

Wiederverwendbarkeit: Fähigkeit der Software, einzelne Komponenten der Software zur Schaffung anderer Software nutzen zu können,

Effizienz, Leistungsfähigkeit (efficiency): Fähigkeit der Software, gut mit den Hardwareressourcen und Betriebssystemdiensten umzugehen,

Modularität: Eigenschaft der Software, in mehr oder weniger autonome Komponenten zerlegbar zu sein bzw. aus autonomen Komponenten zu bestehen,

Kontinuität: Eigenschaft der Software, dass kleinere Veränderungen der Anforderungen nicht zu einer gewaltigen Änderung der Software führen müssen, d. h. kleine Änderungen nur wenige Module betreffen,

Benutzbarkeit (usability): Eignung der Software für den Benutzer, dessen Funktionen, die Bedienung und Handhabbarkeit der Software zu erlernen und deren Meldungen und Ergebnisse zu interpretieren,

Funktionsabdeckung, Funktionalität (functionality): Eignung der Software, seine spezifizierten Funktionen entsprechend den gegebenen Erfordernissen auszuführen,

Instandsetzbarkeit, Pflegbarkeit (maintainability): Eignung der Software, die Erkennung von Fehlerursachen und die Beseitigung von Fehler zu gewährleisten,

Portabilität (portability): Eignung der Software zum Einsatz in unterschiedlichen vorgegebenen Hardware- oder Softwaresystemen,

Sicherheit: Eigenschaft der Software, frei von Gefährdungen zu sein,

Verknüpfbarkeit: Eigenschaft des Programms mit anderen Softwaresystemen verbunden zu werden.

Darüber hinaus gibt es jedoch auch für Dokumente, die im Zusammenhang mit einer Software existieren, eine Reihe von Qualitätsmerkmalen [Glöe98], [DGQ98], [DGQ86], DIN 66270 Qualität der Dokumente/Dokumentation:

Änderbarkeit: Eignung von Dokumenten zur Ermittlung aller von einer Änderung betroffenen Dokumententeilen und zur Durchführung der Änderung,

Aktualität: Übereinstimmung der Beschreibung eines Programms in Dokumenten mit dem jeweils geltenden Zustand des Programms,

Eindeutigkeit: Eignung von Dokumenten zur unmissverständlichen Vermittlung der gleichen Information an den Leser,

Identifizierbarkeit: Eindeutige Ansprechbarkeit der Teile von Dokumenten, die Angaben zu einem abgegrenzten Sachverhalt enthalten,

Normenkonformität: Erfüllung der für die Erstellung von Dokumenten geltenden Vorschriften und Normen,

Verständlichkeit: Eignung von Dokumenten zur erfolgreichen Vermittlung der darin enthaltenen Information an einen sachkundigen Leser,

Vollständigkeit: Vorhandensein der für den Zweck der Dokumente notwendigen und hinreichenden Information,

Widerspruchsfreiheit: Nichtvorhandensein sich entgegenstehender Aussagen in Dokumenten.

Vertiefende Informationen zu diesem Thema können den Normen:

- DIN 66272 - Softwarequalitätsmerkmale,
- DIN ISO 8402 - Qualitätssicherung,
- DIN 66230 - Benutzerdokumentation,
- IEC 1508-3 - Functional Safety-Software requirements,
- DIN V VDE 0801 - Grundsätze für Rechner mit Sicherheitsaufgaben

entnommen werden [Habl98].

## 1.3 Natürliche Sprache als Beschreibungsmittel

Die Zusammenstellung der Qualitätsparameter für Software und ihre Dokumente hat gezeigt, dass Spezifikationen gleichzeitig kompakt, vollständig, konsistent, genau und eindeutig sein müssen. Gemäß der vorherrschenden Meinung, wird die natürliche Sprache jedoch als ungeeignet als Spezifikationsmittel betrachtet. Dies liegt an einigen Eigenarten [Stru91]:

- Die natürliche Sprache ist mehrdeutig („*die Bank*“ – ist das Geldinstitut oder die Sitzgelegenheit gemeint?),
- natürlichsprachliche Aussagen sind oftmals vage („*etwas mehr*“ – wie viel ist das?),
- innerhalb eines Textes besteht die Möglichkeit der elliptischen Verkürzung (einige Aussagen beziehen sich auf vorherige Aussagen, kennt man diese nicht, können Missverständnisse auftreten),
- der metaphorische Sprachgebrauch oder poetische Sprachausdrücke können zu Missverständnissen führen,
- die menschliche Sprache entwickelt sich ständig weiter und ist nicht konstant (neue Wörter und Redewendungen werden geschaffen).

Ein formales verbales Beschreibungsmittel wie die Sicherheitsfachsprache muss diese Nachteile umgehen oder durch Restriktionen beseitigen. Der Gebrauch einer formalen Spezifikationssprache zwingt den Spezifizierer, genau festzulegen, welche Information gegeben werden sollen und welche nicht. Dieses Vorgehen erscheint sicher sehr streng, eventuell auch unkomfortabel, dennoch erhält man dadurch qualitativ viel bessere Beschreibungen als mit natürlicher Sprache [Hogr89].

Ein weiteres Problem der Spezifikation mit natürlicher Sprache besteht in einer unzureichenden Strukturierungsmöglichkeit. Dadurch passiert es, dass oft ungewollt zwischen Abstraktionsebenen gewechselt wird.

Sommerville fasst dies in drei Typen von Problemen zusammen:

1. Mangel an Klarheit,
2. Durcheinander der Anforderungen,
3. Verschmelzung von Anforderungen [Somm97].

## 1.4 Stand der Technik zur Erhöhung der Softwarequalität

Um die Sicherheit eines technischen Systems zu erhöhen, haben sich unterschiedliche Maßnahmen herausgebildet. Zu diesen gehören:

1. Konstruktive Gestaltung des Systems (z. B. der Einsatz von fail-safe-Konstruktionen bzw. sicheren Bauteilen oder die Reduzierung von Stressfaktoren durch Überdimensionierung von Bauteilen),
2. Einsatz von technischen Schutzeinrichtungen (Aufbau redundanter Systeme, Einsatz von fehlererkennenden bzw. -tolerierenden Systemen),
3. Durchsetzung organisatorischer Maßnahmen (regelmäßige Inspektion der Systeme, fachgerechte Schulung des Personals).

Die vorgestellten Handlungen sind sicherlich nicht vollständig, jedoch stellt die gezeigte Reihenfolge bereits eine Priorisierung der Maßnahmen dar. Das oberste Ziel muss es sein, das System von sich aus ‚eigensicher' zu gestalten. Erst wenn diese Maßnahmen ausgeschöpft sind, sollten zusätzliche Schutzeinrichtungen integriert werden, die die Funktionsfähigkeit des Systems überwachen und eventuelle Fehler aufdecken und melden sollen. Sollten auch diese Maßnahmen nicht zu einer ausreichenden Verringerung des Restrisikos führen, muss dieses durch weitere nichttechnische Maßnahmen unter die noch akzeptable Grenze (Grenzrisiko) gesenkt werden.

Auch im Bereich der Softwaretechnik gibt es einige - sowohl technische als auch organisatorische - Maßnahmen, um die Korrektheit und Sicherheit und somit auch die Qualität der betrachteten Software zu erhöhen. Einige davon sollen anschließend kurz erläutert werden. Ebenso wie bei den technischen Systemen lassen sich die Maßnahmen in unterschiedliche Bereiche unterteilen. An dieser Stelle wird eine Einteilung hinsichtlich der Einordnung der Maßnahmen in den Softwareentstehungsprozess vorgenommen.

1. Maßnahmen im Vorfeld und Umfeld der Softwareerstellung,
2. Maßnahmen während der Softwareerstellung,
3. Maßnahmen im Anschluss an die Softwareerstellung.

Zu den unter 1. genannten Maßnahmen im Vor- und Umfeld der Softwareerstellung gehören planerische, administrative und organisatorischen Maßnahmen wie z. B. die Einführung von Qualitätssicherungssystemen, Normen und Richtlinien für die Softwareerstellung, die Durchführung von Audits sowie die geeignete Ausbildung und Motivation der Mitarbeiter [DGQ86], [Balz96] und [Balz98].

Während der eigentlichen Erstellung der Software können geeignete Hilfsmittel, Werkzeuge, Methoden und Sprachen eine technische Unterstützung geben und so zu einer Verbesserung der Softwarequalität beitragen.

Aber auch nach der Erstellung der Software gibt es weiterhin genügend Möglichkeiten, die Korrektheit der Software zur überprüfen und sicherzustellen. Hierzu gibt es wiederum geeignete technische Maßnahmen zur Prüfung der Software, z. B. prüfende oder auch analytische Verfahren.

Diese drei genannten Bereiche werden in den folgenden Abschnitten noch weiter vertieft.

Neben den bereits angesprochenen organisatorischen Maßnahmen zur Erhöhung der Softwarequalität haben sich im Bereich der ‚technischen' Maßnahmen in den letzten Jahren zwei Hauptströmungen herausgebildet. Ein Ansatz besteht darin, den Nachweis der geforderten Eigenschaften am bereits erstellten Steuerungsprogramm durch Tests oder besser durch Verifikation zu erbringen (siehe Abbildung 4). Eine andere Möglichkeit besteht darin, direkt und automatisch aus einer gegebenen Spezifikation ein Steuerungsprogramm zu generieren, das exakt den gestellten Anforderungen entspricht.

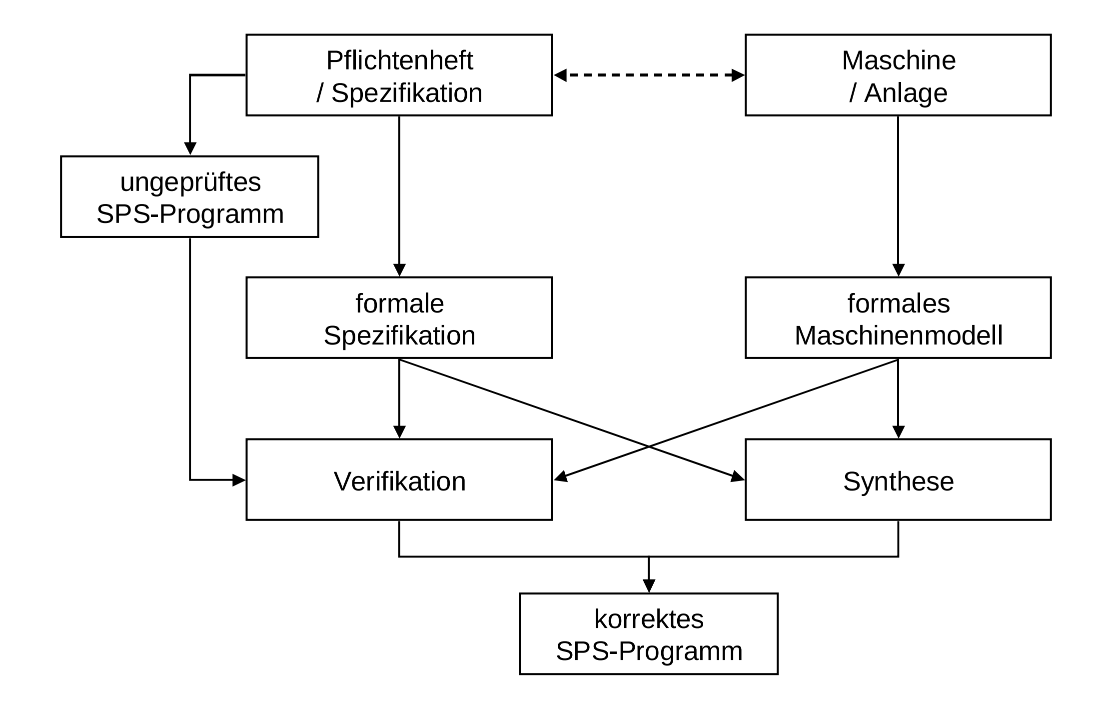



*Abbildung 4 - Vergleich von Synthese und Verifikation*

Heute sind mehr oder weniger leistungsfähige Werkzeuge zur Verifikation vorhanden, zukünftig liegen größere Potenziale aber in der direkten Synthese von Steuerungsprogrammen aus der gegebenen Aufgabenstellung. Beiden Methoden (Synthese, Verifikation) ist die Notwendigkeit einer korrekten, vollständigen und widerspruchsfreien Darstellung der geforderten Programmeigenschaften gemeinsam. Deshalb liegt ein entscheidender Ansatz zur Verbesserung der Softwarequalität in der Anwendung formaler Methoden in der Darstellungs-, Design-, Spezifikations- und Entwicklungsphase der Software.

Ziel der in dieser Arbeit vorgestellten Sicherheitsfachsprache ist es, ein geeignetes Werkzeug für beide Vorgehensweisen zu sein. Wie in der Abbildung 4 zu erkennen ist, stellt die formale Spezifikation als direkte Repräsentation des Pflichtenheftes eine Grundlage für beide Vorgehensweisen dar und ist somit ein zentraler Bestandteil aller weiteren Bemühungen um eine korrekte Software.

### *1.4.1 Maßnahmen im Vorfeld der Softwareentwicklung*

Wie bereits angedeutet, gibt es bereits im Vorfeld der Erstellung eines Programms eine Reihe von Möglichkeiten, Einfluss auf die Qualität des späteren Produktes, also der Software, zu nehmen. An dieser Stelle sollen zwei Maßnahmen in diesem Bereich kurz vorgestellt werden: die Einführung von Qualitätsmanagement-Prozessen und die Arbeit gemäß Vorgehensmodellen.

#### *1.4.1.1 Qualitätsmanagement-Prozesse*

In den letzten Jahren haben sich verschiedene spezielle Formen von Qualitätsmanagement-Prozesse herausgebildet, die sich aufgrund ihres Konzeptansatzes, ihrer Zielstellung, ihrer Durchführung und ihres Detaillierungsgrades unterscheiden. Zusätzlich sind durch den Einsatz in verschiedenen technischen Branchen weitere Unterscheidungsmerkmale gegeben. Allen gemeinsam ist ihr Ziel, die Prozessqualität zu verbessern.

Durch die Norm ISO 9000 und die mit ihr verbundenen Normen wird das Verhältnis von Auftraggebern und Lieferanten in einem allgemeinen, übergeordneten organisatorischen Rahmen zur Qualitätssicherung festgelegt [Balz98]. Dabei zielt die Normengruppe ISO 900*x* auf die Verbesserung der industriellen Produktion ab. Sie ist in den vergangenen Jahren zu einem werbewirksamen de-facto-Qualitätssymbol der Industrie geworden. Die ISO 900x-Normen legen allgemeingültigen Anforderungen an die Aufbau- und Ablauforganisation von Unternehmen fest, sie definieren Dokumente und deren Inhalte. Durch sie wird die Regelung von Zuständigkeiten, Verantwortlichkeiten, Befugnissen gefördert.

Die ISO 9000-1 enthält eine allgemeine Einführung und einen Leitfaden zur Auswahl und Anwendung der weiteren Normen, ISO 9001 beschreibt Modelle zur Darlegung der Qualitätssicherung in Design und Entwicklung, Produktion, Montage und Kundendienst. Die ISO 9002 definiert Modelle zur Darlegung der Qualitätssicherung in Produktion und Montage, ISO 9003 beschreibt die Qualitätssicherung in der Endprüfung. Herauszuheben ist die Teilnorm ISO 9000-3, sie stellt eine Richtlinie zur Anwendung der ISO 9001 für die Entwicklung, Lieferung und Wartung von Software dar. Der Nachweis der Konformität eines Unternehmens zur ISO 900*x* erfolgt durch eine externe Zertifizierung.

Das „Total Quality Management“ (TQM) kann man als eine Erweiterung der ISO 900*x* ansehen. TQM wird definiert als eine „auf der Mitwirkung aller ihrer Mitglieder basierende Führungsmethode einer Organisation, die Qualität in den Mittelpunkt stellt und durch Zufriedenheit der Kunden auf langfristigen Geschäftserfolg sowie auf Nutzen für die Mitglieder der Organisation und für die Gesellschaft zielt“ [ISO 8402]. TQM ist also ein umfassendes Konzept, das das gesamte Unternehmen einbezieht, das die Bedürfnisse des Kunden und die Kundenorientierung in den Mittelpunkt stellt.

Während die ISO 900*x* überwiegend die technischen Aspekte der Arbeit und der Qualitätssicherung behandelt und in diesem Zusammenhang als relativ statisch bezeichnet werden muss, betont TQM die Zusammenarbeit der Mitarbeiter zum Zweck der Erreichung der Qualitätsziele als einen ständigen, kontinuierlichen Prozess.

Das Capability Maturity Model (CMM), das 1987 vom Software Engineering Institut der Carnegie Mellon Universität entwickelt wurde (siehe [DGQ98] und [Balz98]), ist rein software-orientiert und bewertet den aktuellen Stand der Qualitätssicherungsmaßnahmen in einem Unternehmen mit fünf Stufen (Levels).

- Level 1: „Initial“, keine stabilen, wiederholbaren Prozesse, Erfolge werden durch Einzelpersönlichkeiten erzielt,
- Level 2: „Repeatable“, es sind wiederholbare Prozesse und etablierte Managementmaßnahmen vorhanden,
- Level 3: „Defined“, Standardprozesse sind etabliert,
- Level 4: „Managed“, Prozesse werden bewusst gesteuert, Qualitätsziele sind quantifizierbar und werden durch standardisierte Verfahren geprüft,
- Level 5: „Optimizing“, die Prozesse werden weiterhin verbessert.

Die Beurteilung über die Erreichung eines bestimmten Levels erfolgt durch Fragebögen mit reinen Ja/Nein-Fragen in so genannten Assessments. Bei der Bootstrap-Methode, einem Sonderfall des CMM, erfolgt die Bewertung durch eine externe Begutachtung statt durch Eigenbewertung.

Auf Initiative der ISO und der IEC wurde 1993 SPICE (Software Process Improvement and Capability Determination) entwickelt. Hier finden sich ähnliche Reifegradstufen wie bei CMM, jedoch wird auch das Prozessumfeld bei der Beurteilung der Prozesse berücksichtigt. SPICE ermutigt die Anwender zur Selbstbewertung, diese erfolgt in einer fünfstufigen Skala, nicht mehr nur durch Ja/Nein-Fragen. In Zukunft sollen Ansätze wie ISO 9000 und CMM in SPICE integriert werden.

Inwieweit solch komplexe Maßnahmen in eine betriebliche Struktur integriert werden können (letztendlich müssen alle Mitarbeiter beteiligt sein), hängt natürlich von individuellen Einflüssen ab. Jedoch lassen sich auch mit der Auswahl einzelner kleiner Komponenten bereits gute Ergebnisse bei der Fehlervermeidung erzielen. In [Mont00] werden drei grundsätzliche Schritte vorgestellt: Prävention, Test, Fehlerbehandlung. Mit einer Vereinfachung der Arbeitsweise, der Protokollierung alter Fehler, durch die Durchführung von Inspektionen (bzw. Audits) oder auch durch die Mehrfachentwicklung von Komponenten lässt sich die Qualität von Programmen bereits rapide verbessern.

#### *1.4.1.2 Vorgehensmodelle*

Zur Reduzierung der Komplexität bei der Lösung von Entwicklungsaufgaben wurden Vorgehensmodelle (auch Phasenmodelle) entwickelt. Diese sollen eine Strukturierung und somit eine Vereinfachung der Aufgabenstellung in einzelne Teilaufgaben und Entwicklungsabschnitte ermöglichen.

Ausgehend von diesem Hauptziel lassen sich weitere Teilziele formulieren:

- Unterstützung des Projektmanagements durch Segmentierung des gesamten Lebenszyklusses in kleinere Entwicklungsschritte,

- Vorgabe eines ablauforganisatorischen Rahmens mit Festlegung von Schnittstellen zwischen den einzelnen Projektphasen,
- Definition exakter Ziele in den einzelnen Projektphasen,
- Unterstützung der Kommunikation durch Prüfungen und Übergabe der Ergebnisse von einer Projektphase in die folgende.

Beispiele für gebräuchliche Vorgehensmodelle sind: (siehe [DGQ98] und [Balz98])

- Code-and-Fix (Programmiere-und-Repariere-)Modell
- sequentielles Modell (Stagewise Model),
- Schleifen-, Wasserfallmodell,
- Prototyping-Modell
- Sichtenmodell,
- Spiralmodell,
- V-Modell,
- u. a.

Allen Modellen gemeinsam ist die Unterteilung des Projektes in einzelne Phasen. Die Aufgabe wird jedoch nicht nur zeitlich zerlegt, es erfolgt auch eine Gliederung in einzelne Teilbereiche des Produktes, so dass man von einer vertikalen und einer horizontalen Projektorganisation sprechen kann. Dabei lässt sich bei den einzelnen Modellansätzen sogar eine weitgehende Übereinstimmung in der Gliederungsmethodik und der Bezeichnung der Phasen erkennen, die Unterschiede treten erst bei der Betrachtung der Verbindungen und Schnittstellen zwischen den einzelnen Phasen hervor.

Am Beispiel des weit verbreiteten V-Modells soll der Inhalt eines solchen Vorgehensmodells kurz erklärt werden. Der Entwicklungszyklus eines Produktes lässt sich vereinfacht in die folgenden sieben Phasen einteilen:

1. Planung, Zielvorgabe durch den Auftraggeber, hier entsteht das Lastenheft,
2. Problemanalyse, -definition, Problemspezifikation, hier entsteht das Pflichtenheft,
3. Entwurf des Systems, Systemspezifikation, Zergliederung in Komponenten,
4. Implementierung und Realisierung der Einzelmodule, Test, Dokumentation,
5. Vervollständigung des Gesamtsystems, Funktions- und Leistungsprüfung, hier ist das funktionsfähige System fertig,
6. Installation und Abnahme durch den Auftraggeber,
7. Betrieb und Wartung.

Im V-Modell werden diese Schritte ebenfalls durchlaufen, die grafische Anordnung der einzelnen Schritte erfolgt in Form eines V (siehe Abbildung 5). Dies geschieht, um die inhaltlichen Zusammenhänge der Einzelphasen des linken Astes mit denen des rechten Astes gegenüberstellen zu können. Hierbei werden in einer zusammen-

hängenden Darstellung die Entwurfs- und Implementierungsschritte mit den jeweiligen Testprozeduren und -verfahren verknüpft.

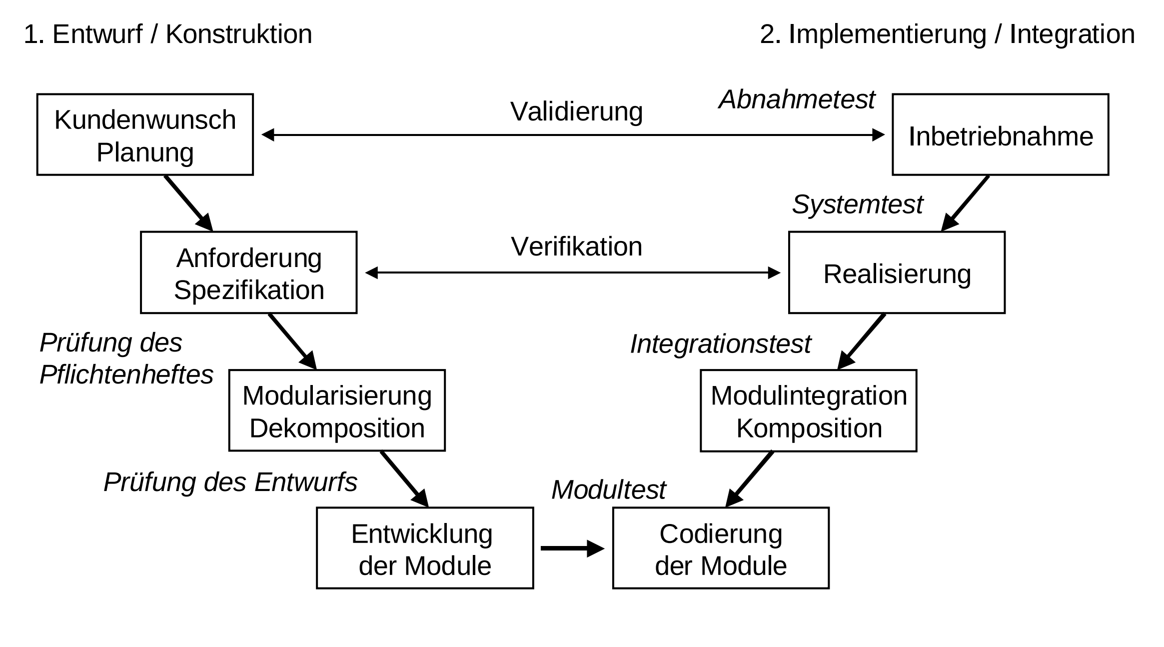



*Abbildung 5 - Das V-Modell*

Das V-Modell ist im Besonderen für die Bewältigung großer Projekte geeignet.

### *1.4.2 Maßnahmen während der Softwareentwicklung*

Ungeachtet der im vorherigen Abschnitt dargestellten Möglichkeiten für die Erhöhung der Softwarequalität durch vorbereitende und begleitende Maßnahmen, gibt es natürlich auch beim eigentlichen Vorgang der Softwareerstellung spezifische fehlervermeidende Maßnahmen. Dabei bedient man sich beispielsweise ähnlicher Ansätze, wie bei der Hardwaregestaltung sicherheitskritischer Systeme. D. Hablawetz beschreibt in [Habl98] die diversitäre Programmierung eines Programms als eine Lösung. Hierbei wird durch unterschiedliche Programmierteams, möglicherweise auch mit unterschiedlichen Programmiersprachen, ein und dasselbe Problem gelöst. Anschließend lässt man die erstellten Programme parallel laufen und vergleicht die gezeigten Reaktionen.

#### *1.4.2.1 Spezifikationsmethoden*

Bereits im Abschnitt 1.1.2 wurde deutlich gemacht, dass die aktuell verwendeten Methoden und Werkzeuge für die Erstellung von Steuerungsprogrammen nicht mit der Komplexität der eingesetzten Hardware übereinstimmen. Dieser Umstand kann nur durch den konsequenten Einsatz formaler Methoden, die auf den Anwendungsfall angepasst sind, behoben werden. Bereits in [Wehr96] wird dieser Einsatz formaler Methoden für den Entwurf verteilter reaktiver Systeme gefordert, besonders in der [IEC1508] wird eine Definition der Anforderungen mit Temporaler Logik vorgeschlagen. Dieser Trend lässt sich in vielen Forschungsprojekten und den zugehörigen Veröffentlichungen nachweisen. So gibt es bereits seit fast 20 Jahren Standard-Spezifikationssprachen für einzelne Einsatzgebiete, die Ursprünge hierfür sind im Be-

reich der Kommunikationsdienste zu finden. Der folgende Abschnitt soll einige der aktuellen Methoden und Werkzeuge und die dabei verwendeten Spezifikationstechniken darstellen.

Diese lassen sich in drei Kategorien zusammenfassen:

1. Grafische Notationen,
2. Mathematische Spezifikationen,
3. Strukturierte natürliche Sprache.

***Estelle***, ***LOTOS*** (Language of Temporal Ordering Specification, ISO/IEC 8807) und ***SDL*** (Specification and Description Language) sind ereignisbasierte Methoden, die für die Spezifikation verteilter Systeme und Kommunikationsprotokolle und -dienste zum Einsatz kommen ([Hogr89], [Frap01], [Reck91]). SDL erlaubt speziell die Modellierung eines reaktiven Systems als ein aus mehreren Komponenten aufgebautes System. Die einzelnen Systeme werden durch einen endlichen Automaten beschrieben und kommunizieren asynchron miteinander [Lewe97].

***Z*** (1988 von Spivey entwickelt), ***B*** und ***VDM*** (Jones 1986) sind zustandsbasierte Methoden, die besonders für die Beschreibung des Verhaltens von sequentiellen Systemen geeignet sind ([Clar96], [Frap01]).

Die hauptsächlichen Einsatzgebiete von ***CSP*** (Hoare 1985), ***Statecharts*** (Harel 1987) und der ***Temporalen Logik*** (Manna, Pnueli 1991) ist die Beschreibung des Verhaltens von nebenläufigen und parallelen reaktiven Systemen ([Clar96], [Mann92], [Mann95]). Statecharts modellieren reaktive Systeme als System kommunizierender und hierarchisch organisierter endlicher Automaten, Temporale Logiken erlauben die Formulierung von Anforderungen durch modallogische Formeln [Lewe97].

***VHDL*** und ***Verilog*** sind Notationen, die zur Beschreibung von digitalen Schaltkreisen eingesetzt werden ([Lewe97], [Frap01]).

***SREM*** (Software Requirement Engineering Method) [DGQ92] ist ein Werkzeug zur Übersetzung funktionaler Anforderungen in detaillierte und maschinell nachprüfbare Spezifikationen. SREM enthält drei Komponenten:

- ***RSL*** (requirement statement language): Formulierung von Anforderungen in lesbarem Englisch (mit strenger Syntax), Überprüfung auf Widerspruchsfreiheit und Vollständigkeit möglich,
- ***REVS*** (requirement engineering and validation system): Unterstützung bei der Übersetzung der RSL-statements, Speicherung, Analyse, Simulation,
- ***R-net*** (requirements network): grafische Darstellungsform.

Das Einsatzgebiet von SREM liegt bei der Entwicklung von komplexen Realtime-Systeme mit Parallelverarbeitung [DGQ92].

Zur Unterstützung großer Projekte wurden strukturierte Methoden entwickelt. In diesem Zusammenhang sollen ***SA*** (Structured Analysis), ***SADT*** (Structures Analysis and Design Technique) und ***SA/RT*** erwähnt werden ([Reck91], [Part98]). In den letz-

ten Jahren wurden neue objektorientierte Methoden entwickelt: ***OOA***, ***OMT*** und die ***UML*** (Unified modeling language).

Die ***UML*** ([Oest98], [Booc99], [Frap01]) hat sich in der jüngsten Vergangenheit zu einer Standard-Spezifikationssprache für objektorientierte Software (vor allem im Bereich der Informationstechnik) herausgebildet. Sie ist sowohl eine Visualisierungs-, Dokumentations- und Konstruktionssprache und besteht aus verschiedenen Diagrammen mit verschiedenen grafischen Elementen, die je nach spezieller Aufgabe eingesetzt werden können. Bei vielen Tools, die UML benutzen ist eine automatisierte Übersetzung der Spezifikation in C++ oder Java möglich. [Braa01] und [West01] stellen darüber hinaus mit ***ODEMA*** eine Methode zur Softwarespezifikation für die Produktionsautomatisierung vor. Die UML dient dabei als Notationswerkzeug, die einzelnen Diagrammtypen werden in ihrer Semantik soweit präzisiert, dass sie den Aktivitäten eines Entwicklungsprozesses zugeordnet werden können. So wird z. B. das Verteilungs- (Deployment-) Diagramm für die Darstellung der Schichten der Automatisierungspyramide verwendet. In [Xu01] wird die Erweiterung der UML zu UML-RT (Real-Time) beschrieben, die als geeignete Modellierungssprache für die Maschinenmodellierung dienen soll. Aus dem Bereich der Automobilindustrie kommt ein Vorschlag für die Erweiterung der Standard-UML um automobiltechnische Aspekte zur Automotive-UML [Hofm00].

Als aktuelle Entwicklung im Bereich der Spezifikationstechniken soll auf ***XML*** (Extensible Markup Language) verwiesen werden [Birk00], an die gerade im Bereich des Anforderungsmanagements der Webtechnologien große Erwartungen gestellt werden. Die Besonderheit der XML besteht in der Trennung von Inhalt, Struktur und Layout der zu entwickelnden Applikation.

Die Darstellung von Programmanforderungen mit Temporaler Logik hat sich bereits als sinnvoll erwiesen und wird auch so gefordert [IEC1508]. Da diese jedoch schwierig zu verstehen und zu benutzen ist, werden alternative Darstellungsmöglichkeiten gesucht, mit denen ein leichterer Zugang möglich wird. In [Dill94] wird ***GIL*** (Graphical Interval Logic) vorgestellt. GIL ist die Basis für ein Toolset zur Unterstützung der formalen Spezifikation und Verifikation von Softwaresystemen mit Hilfe einer so genannten visuellen Temporalen Logik. Die Darstellung der Anforderungen erfolgt in Form von Timing Diagrammen, mit denen Hardwaredesigner und Softwareingenieure vertraut sind. Man erhält somit eine intuitive grafische Darstellung. Mit GILED, dem zugehörigen Editor, werden grafisch Intervalle definiert, in denen Eigenschaften erfüllt sein müssen. Die TL-Operatoren (Eventually, Until und Henceforth) sowie logische Operatoren (UND, ODER, Implikation, Äquivalenz, Negation) können somit ausgedrückt werden. Das gesamte Tool-Set enthält darüber hinaus einen „Proof Checker“ und einen „Model-Generator“, so dass eine grafische Überprüfung und ein Beweis von Systemeigenschaften möglich sind.

Die Anzahl der bekannten Entwicklungswerkzeuge, die natürliche Sprache zur Formulierung der Anforderungen nutzen, ist gering: ***AMADEUS*** ([Blac87]) ***ALECSI*** ([Cauv91]). Von H. Dalianis wird in [Dali95a] und [Dali95b] ein Spezifikations- und Validierungswerkzeug beschrieben, das zur Entfernung von Redundanzen in Texten verwendet werden kann. Mit ***VINST*** (Visual and Natural language specification tool) erfolgt die Spezifikation mit der Visual Language (VL) und einer begrenzten Natürlichen Sprache (NL). Ziel ist die Generierung eines redundanzfreien natürlichsprachlichen Textes, speziell für die Funktionen von Telekommunikationsdiensten. Eine an-

schließende Übersetzung nach LOXY (First Order Logic extended with Time), einer speziellen Form der Temporalen Logik ist möglich.

Im deutschsprachigen Raum ist neben den Arbeiten von [Hölz96] und [Hölz98a], die so genannte Schablonen für die natürlichsprachliche Spezifikation benutzen (siehe Abschnitt 1.4.3.2), die parallele Darstellung von formalen Anforderungen mit temporallogischen Formeln und natürlichsprachlichen Darstellungen durch [Bits02] zu nennen. Hier erfolgt die Darstellung der Spezifikation mit „Safety-Patterns“, die vordefinierte Satzstrukturen bereitstellen und durch den Anwender zu vervollständigen sind.

Einen etwas anderen Weg gehen die beiden folgenden Tools, deren Ziel es ist, aus einem Text die wesentlichen Aussagen und Anforderungen herauszufinden. Diese müssen meist aus einer großen Menge an Text gefunden werden, der von Kunden und Anwendern zusammengestellt wurde. Das Programm ***Findphrases*** dient zum Auffinden wiederholter Phrasen in einem willkürlichen Text [Agui90]. Hier geht es darum, die ursprüngliche Aufgabenstellung zu präzisieren und zu abstrahieren. ***Abstfinder*** [Gold94] dient ebenfalls zum Auffinden der Abstraktion innerhalb eines natürlichsprachlichen englischen Textes.

Für den speziellen Bereich der Software für Steuerungssysteme gibt es eine sehr große Auswahl von Beschreibungsmitteln. In [Polk92] werden diese dargestellt und kategorisiert. Man unterscheidet demnach prozedurale und logik-orientierte Sprachen. Eine weitere Unterteilung wird jeweils noch in grafische und textorientierte Beschreibungsmittel getroffen, so dass insgesamt vier Kategorien unterschieden werden.

Prozedural grafische Beschreibung der Funktion:

- Ablaufdiagramm,
- Bewegungsdiagramm,
- Weg-Schritt-Diagramm,
- Weg-Zeit-Diagramm,
- Zustandsdiagramm,
- Flussdiagramm, Programmablaufplan (DIN 66001, DIN 66262),
- Funktionsplan (DIN 40719/6), [Fein97],
- Struktogramm nach Nassi/Sheidermann (DIN 66261),
- Automatengraf [Heri92],
- Petrinetz [Reis85],
- SPS-Programmierung nach DIN 61131-3: Kontaktplan KOP, Funktionsbausteinsprache FBS, grafische Ablaufsprache AS

Prozedural textuell:

- Pseudocode,
- SPS-Programmierung nach DIN 61131-3: Anweisungsliste AWL, Strukturierter Text ST, textuelle Ablaufsprache AS

Logik-orientiert grafisch:

- Schaltnetz,
- Logikplan,
- Schalttabelle,
- Entscheidungstabelle nach DIN 66241,

Logik-orientiert textuell:

- algebraische Spezifikationen,
- logik-orientierte Sprachen.

In [Chou98b] sind mehr als 40 Beschreibungssprachen zusammengetragen, klassifiziert und bewertet worden.

Vom selben Autor wird in [Chou98a] die Spezifikation einer Steuerung mit einer TL-ähnlichen Sprache beschrieben. Hierbei werden die Anforderungen mit so genannten Verbots- und Gebotsoperationen verbal ausgedrückt („ist verboten, ist verboten nach, ist verboten vor, soll, soll vor, soll nach“).

Als weiteres grafisches Werkzeug zur formalen Spezifikation von Steuerungssystemen berichten [Jörn96] und [Frey98] über die Steuerungsentwicklung mit IPN (Interpretierte Petri-Netze nach König und Quäck). Es handelt sich hierbei um den systematischen und hierarchischen Entwurf von Steuerungen mit dem Tool ***Netmate***.

### *1.4.2.2 Programmierregeln*

Als ein wichtiges Mittel zur direkten und schnellen Verbesserung der Softwarequalität haben sich Entwurfs- und Programmierregeln bewährt, die grob in drei Bereiche aufgeteilt werden können:

- Ausführliche Dokumentation (Kommentare im Code, Referenzlisten, Schriftbild, Übersichtlichkeit, Änderungen, Übersichtlichkeit),
- Defensive Programmierung (keine Tricks, geringe Schachtelungstiefe von Modulen)
- Vollständigkeit des Codes (alle Variablen deklarieren und initialisieren, vollständige Fallunterscheidungen, Wertebereichs- und Plausibilitätsüberprüfungen, Abbruchbedingungen in Schleifen)

In [Rein99] und [Balz96] sind einige dieser leicht anzuwendenden und wirksamen Maßnahmen aufgeführt.

### *1.4.2.3 Softwaresynthese*

Schon seit längerer Zeit gibt es das Bestreben, Software und auch speziell Steuerungsprogramme automatisch aus einer vorgegebenen Aufgabenstellung heraus zu generieren. Parallelen hierzu gibt es aus dem Bereich der Regelungstechnik, wo es möglich ist, bei genügend genauer Kenntnis der Regelstrecke und der Vorgabe eines Regelziels den dazugehörigen Regler zu berechnen.

Bereits 1996 wurde durch K. Winkelmann ein Patent eingereicht (WO 96/32667), das ein Verfahren zur Generierung einer Steuerung aus einer anwendungsnahen formalen Spezifikation beschreibt. Ziel ist die Erzeugung einer Steuerung mit spezifizierten Sicherheitseigenschaften, wobei der Generierungsprozess die Einhaltung der Sicherheitsbedingungen garantiert. Mit einer nicht näher ausgeführten Spezifikationssprache ***CSLxt*** kann die Funktion durch die Angabe bedingter Zielzustände beschrieben werden. Diese enthält die Beschreibung von Eingangszuständen, Ausgangszuständen, internen Zuständen sowie die Definition von Sicherheitseigenschaften. In einem anschließenden Minimax-Verfahren wurde die Steuerung generiert.

K. Lemmer et al. beschreiben in [Lemm95], wie dieser Synthesevorgang durchgeführt werden kann. Der vorgestellte Ansatz wird auch in [Hani96] und [Hani98] beschrieben, wobei Hanisch, Rausch, Thieme und Lüder (Uni Magdeburg) eine automatische Codegenerierung auf der Basis von Netzmodellen präsentieren. Durch Modellierung der ungesteuerten Strecke (der zu steuernden Maschine bzw. Anlage) mit Netzstrukturen erfolgt eine Beschreibung des physikalisch möglichen Verhaltens dieser Maschine. Mit Hilfe der Spezifikation verbotener Zustände (Fakten) sowie der Beschreibung geforderter Abläufe (Start und Zielstellen für Marken) lässt sich die zugehörige Steuerung berechnen. Die vorgestellte Methodik wurde in den Tools ***SAFECODE*** und ***MASC*** umgesetzt und erlaubte die automatische Codeerzeugung in IEC 1131-3 konforme Sprachen.

Zwei weitere Forschungsprojekte sollen noch erwähnt sein. Innerhalb des Schwerpunktprogramms „Integration von Techniken der Softwarespezifikation für ingenieurwissenschaftliche Anwendungen“ (SPP 1064) beschäftigt sich das Teilprojekt „***SpeciMen***“ mit der Integration von Spezifikations- und Modellierungstechniken bei der Modellsynthese im Steuerungsentwurf ([http://mathsrv.ku-eichstaett.de/MGF/informatik/Projekte/Specimen/specimen.html], KU Eichstätt (Prof. Desel), Uni Halle (Prof. Hanisch)). Ziel ist die Entwicklung einer Methodik zur Konstruktion eines integrierten Modells aus gegebenen verschiedenartigen Spezifikationen, so dass das Verhalten des Modells den Spezifikationen genügt und eine Synthese des Modells möglich ist. Als formales Modell werden geeignet definierte Petri-Netze verwendet. Innerhalb desselben Schwerpunktprogramms wird in [Bits02] die Spezifikation von Sicherheitsanforderungen mit so genannten „Safety-Patterns“ vorgeschlagen. Hierbei handelt es sich um einen Katalog fest vorgegebener Sprachkonstrukte, die mit konkreten Programmanforderungen zu vervollständigen sind und eine temporallogische Basis haben.

Das Projekt ***SACRES*** (Safety Critical Real Time Embedded Systems), [www.tni.fr/sacres] befasst sich mit der Entwicklung von zuverlässigen sicherheitskritischen eingebetteten Systemen. Ziel ist die Reduzierung von Designfehlern und eine Verringerung der Entwicklungszeiten und -kosten. Die Spezifikation erfolgt mit SILDEX, Statemate und TDE dem Timing diagram Editor. Eine automatische Codegenerierung sowie eine formale Verifikation durch Model-Checking mit SVE (Software Verification Environment) gehören ebenfalls zum Umfang der Forschungen.

Tools für die automatische Codegenerierung stehen auch bereits für den industriellen Einsatz zur Verfügung. Besonders die Automobilindustrie ist hier ein bevorzugtes Einsatzgebiet, stellt doch der ständige Drang nach innovativen Lösungen bei gleich-

zeitiger Sicherung der Funktion (Stichwort: Rückrufaktionen) eine große treibende Kraft dar. Die Firma dSPACE bietet mit ***TargetLink*** ein Softwarepaket an, mit dem eine Generierung und Optimierung von Steuerungscode, der mit Matlab-Simulink und Stateflow erzeugt wurde, möglich ist [Köst00].

Ebenfalls in diesem Bereich angesiedelt ist ***Rhapsody in MicroC*** [Sche00]. Hiermit lässt sich extrem kompakter Code für Kfz-Steuergeräte mit einer grafischen Programmierumgebung durchgängig von der Spezifikation bis zur Implementierung generieren. Als Besonderheit wird OSEK, ein Standard für eine offene Gerätearchitektur in der Automobilindustrie, unterstützt.

### 1.4.3 Maßnahmen im Anschluss an die Softwareentwicklung

Als abschließendes Kapitel im Zusammenhang mit Wegen zur Erhöhung der Qualität von Software sind diejenigen Maßnahmen zu nennen, die auch noch im Anschluss an die Erstellung der Software ergriffen werden können. Hierbei geht es darum, Fehler und Unzulänglichkeiten der Programme vollständig und sicher zu erkennen, und geeignete Schritte, wie Korrekturen oder eine Neucodierung, einzuleiten. Je nach Wirksamkeit und Abdeckungsgrad unterscheidet man testende und verifizierende Verfahren.

#### 1.4.3.1 Test

Der einfachste Fall der Überprüfung der Korrektheit eines Programms ist der Test. Hierbei unterscheidet man statische Tests, wie Quelltextanalyse, Inspektion und Walkthrough, bei denen der Code visuell von einer Person untersucht und „in Gedanken“ abgearbeitet wird. Bei dynamischen Tests wird das Programm dagegen tatsächlich ausgeführt bzw. simuliert. Man unterscheidet hierbei nochmals in den funktionalen Black-box-Test und den strukturorientierten White-box-Test.

In [Mewe95] wird eine Testumgebung vorgestellt, mit der es möglich ist, Software bereits in frühen Entwicklungsstufen prüfen zu können. Der reale Code wird dabei in eine Umgebung eingebettet, die auf die Ergebnisse des Programms reagiert und aufgrund bestimmter Algorithmen neue Eingangswerte für den zu testenden Code generiert. Diese Systeme sind heute weit verbreitet und auch unter der Bezeichnung „Software in the loop“ (SIL) bekannt.

Als eine Möglichkeit zur Unterstützung der Abnahme von Softwareprodukten enthält die DIN 66285 (siehe [Lind93]) eine detaillierte Beschreibung der Prüfbedingungen für Software. Der Autor führt dabei die Umsetzung der Vorgaben der DIN 66285 in konkrete Handlungsschritte aus. Wesentliche Ziele dieser Darstellung sind:

- Reduktion des Schreib- und Protokollierungsaufwands durch standardisiertes Vorgehen,
- Reduktion des Prüfaufwandes durch produktabhängige Festlegung der Prüftiefe.

Die hersteller- und prüfstellenseitigen Prüfunterlagen werden hier ebenso beschrieben, wie die Ergebnisdokumente und Prüfberichte.

Eine weitere Variante wird in [Grel93] dargestellt. Hier wird durch eine diversitäre Rückwärtsanalyse des erzeugten Programmcodes überprüft, ob das erstellte Programm den ursprünglichen Anforderungen entspricht.

#### *1.4.3.2 Verifikation*

Die im vorherigen Abschnitt erwähnten Tests hinsichtlich der Korrektheit von Programmen müssen aufgrund der Komplexität von Programmen mit einer industriell relevanten Größe immer als unvollkommen betrachtet werden. Als ein weiteres mächtigeres Werkzeug zur Überprüfung der Korrektheit einer Software hat sich deshalb in den vergangenen Jahren die Verifikation herausgebildet. Man unterscheidet hier zwei rechnergestützte Verfahren für die formale Verifikation: das Theorem-Proving und das Model-Checking. Beim Theorem-Proving wird ein Beweisen der Korrektheit aufgrund syntaktischer Formelumwandlungen durch vollständige Induktion bzw. Deduktion geführt. Beim Model-Checking wird der mathematische Beweis durch Überprüfung der formalen Spezifikation an einem Modell des Systems erbracht. Das Modell kann dabei auf verschiedene Arten beschrieben werden, häufig wird eine Darstellung als endlicher Automat verwendet. Zur Beschreibung der Spezifikation werden häufig Formeln der Temporalen Logik genutzt. Die Anfänge dieser Methodik reichen bereits einige Jahre zurück. In [Moon91] wird bereits die Anwendung der Verifikation im Bereich der Chemieindustrie beschrieben. Beispiele für gebräuchliche Model-Checking-Werkzeuge sind ***SMV*** (Carnegie-Mellon University) [McMi92], ***INA*** [Star92], ***PROD*** [Varp95], ***PEP*** [Best95], ***HyTech*** (U. C. Berkeley) [Alur96] und ***Kronos*** (Verimag). Das Model-Checking-Verfahren liefert als Ergebnis der Überprüfung entweder eine Bestätigung (d. h. einen formalen Beweis) für die vollständige Einhaltung der Spezifikation in allen modellierten Zuständen des Systems oder mögliche Gegenbeispiele, in denen eine oder mehrere Anforderungen nicht erfüllt werden.

Nachfolgend werden einige der aktuellen Projekte dargestellt, die nicht nur den Stand der Forschungen auf dem Gebiet der Verifikation von Steuerungsprogrammen, sondern auch die Vielfalt der dabei eingesetzten Werkzeuge zeigen sollen. [Frey00] gibt hierzu einen ausführlichen Überblick über aktuelle Validation- und Verifikationsansätze.

Bereits 1996 wird über eine formale Verifikationsmethode einschließlich eines Werkzeugs berichtet, mit denen die Korrektheit von SPS-Programmen vollautomatisch überprüft werden kann ([Hölz96], [Hölz98a] und [Hölz98b]). Die Spezifikation der Eigenschaften und Annahmen erfolgt mit natürlichsprachlichen Schablonen („IMMER GILT ..., SOLANGE ... GILT ..., NIE GILT ....“), in die mit Programmvariablen gebildete Ausdrücke eingesetzt werden (alle_spindeln_aus == 1). Diese Verifikationsmethode wurde in HiGraph eingebettet und bot die Möglichkeit zur Generierung einer Widerlegungssequenz und zur Darstellung der Variablenzustände für jeden SPS-Zyklus.

In [Canv97] wird das System ***VSE*** (Verification Support Environment) beschrieben, ein Werkzeug zur formalen Spezifikation und Verifikation von sicherheitsrelevanten Softwaresystemen. Dieses entstand im Auftrag des Bundesamtes für Sicherheit in der Informationstechnik und umfasst neben Spezifikations- und Verifikationswerkzeugen auch die zugehörige Entwicklungsmethodik für Systeme mit Sicherheitsver-

antwortung. Die Sicherheitsanforderungen dieser Systeme können formal spezifiziert und deren Einhaltung mathematisch nachgewiesen werden. Anschließend kann Programmcode in Ada oder C generiert werden. Innerhalb des Systems VSE erfolgt die Erstellung eines Sicherheitsmodells sowie eines Funktionsmodells des untersuchten Systems. Die Beschreibung wird mit der Spezifikationssprache VSE-SL vorgenommen.

Kowalewski et al. arbeiten ebenfalls mit einer modellbasierten Verifikation ([Kowa96] und [Kowa01]). Die zu untersuchenden Steuerungsprogramme liegen hierbei in Ablaufsprache (SFC) oder in Anweisungsliste (AWL) vor und werden in Bedingungs-/Ereignis-Systeme (Sreenivas/Krogh) (B/E-Systeme, engl. condition-/event-systems, C-/E-systems) überführt. Ein zusätzliches B/E-System wird für die Modellierung der ungesteuerten Anlage verwendet. Durch Verschmelzung dieser beiden Systeme entsteht ein Modell der gesteuerten Anlage. Durch Erreichbarkeitsanalysen wird nachgewiesen, ob die Spezifikation (dargestellt durch verbotene Zustände) erfüllt wird. Die Besonderheit dieses Ansatzes besteht vor allem in der Ausrichtung auf anlagentechnische Prozesse (Chemietechnik) und den sich daraus ergebenden Besonderheiten bei der Modellierung kontinuierlicher Prozesse. Diese Arbeiten wurden mit dem Tool ***VERDICT*** ([Kowa98], [Tres00a], [Tres00b]) fortgesetzt, bei dem eine Verifikation mit KRONOS und HyTech (auch mit SMV) erfolgt. Die Zielstellung dieses Tools liegt in der Untersuchung diskreter Steuerungen für technische Systeme mit kontinuierlicher Dynamik und hybrider Systeme.

Lesage, Couffin, de Smet, Rossi u. a. [DeSm00], [Lamp99] von Alcatel und der Ecole Normale Superieure (Cachan, Frankreich) überführen innerhalb des Projektes ***VULCAIN*** Steuerungsprogramme, die in den IEC-61131-3 Standard-Programmiersprachen Anweisungsliste [Cane00], Kontaktplan [Ross00a], [Ross00b] oder Ablaufsprache [Lamp00] geschrieben sind, in ein Transitionssystem bzw. direkt in Code, der durch einen Model-Checker (in diesem Fall Cadence-SMV) analysiert werden kann. Die zu überprüfenden Eigenschaften werden hierbei direkt in CTL oder LTL formuliert. Dieser Ansatz verzichtet auf ein Modell der ungesteuerten Strecke, das Verhalten des SPS-Programms wird inklusive des SPS-Zyklus nachgebildet.

Hanisch, Vyatkin (beide Uni Halle) und Starke (HU Berlin) bearbeiten innerhalb des Projektes ***VEDA*** (Verification Environment for Distributed Applications) die Problematik der Modellierung und Verifikation der Abarbeitungssteuerung von Funktionsblöcken nach IEC 61499 mit Signal/Ereignis-Netzen [http://at.iw.uni-halle.de/~valeriy/project/project.html]. Die hier verwendeten Funktionsblöcke nach IEC 61499 unterstützen Formalismen, um verteilte Steuerungssysteme zu spezifizieren. Ausgangspunkt der Verifikation ist also eine Steuerung gemäß IEC 61499, die in ein Modell überführt wird. Zusätzlich erfolgt eine Spezifikation des korrekten bzw. inkorrekten Verhaltens durch CTL-Ausdrücke, die mit einer speziellen Beschreibungssprache - der Timing Diagram Specification Language (TDSL) - erstellt werden. Anschließend erfolgen Untersuchungen am Erreichbarkeitsgraf, mit einem internen Model-Checker oder dem an der HU Berlin entwickeltem SESA.

In [Bits00] wird ein Verfahren vorgestellt, mit dem eine formale Verifikation von Softwarespezifikationen durchgeführt werden kann. Die Verifikation erfolgt dabei anhand von Modellen, die mit industriell üblichen Entwurfswerkzeugen (ASCET-SD und Matlab) erstellt wurden. Diese Modelle werden in SMV-Code überführt, die Spezifikation der Anforderungen erfolgt mit CTL-Formeln.

Zum Abschluss dieses Kapitels seien noch die beiden Forschungsprogramme ***OFFIS*** und ***KORSYS*** genannt, unter deren Federführung weitere Teilprojekte, unter anderem auch im Bereich der formalen Verifikation von Steuerungsprogrammen, durchgeführt werden. Innerhalb von OFFIS [www.offis.uni-oldenburg.de/] werden grafikorientierte Methoden entwickelt, die es den Systementwickler ermöglichen sollen, vorwiegend grafische Beschreibungsmittel sowohl für die Systeme als auch die Spezifikationen zu verwenden. Grafische Methoden wurden gewählt, weil diese leichter zugänglich sind und somit eine höhere Akzeptanz für die Einführung formaler Methoden bieten. Die Systembeschreibungen erfolgen hierbei mit Statemate, die Spezifikationen werden mit symbolischen Zeitdiagrammen ausgeführt, die in Temporale Logik übersetzt werden können. Unter dem Titel KORSYS (Korrekte Software für sicherheitskritische Systeme) werden Verifikationstechniken für eingebettete Systeme weiterentwickelt und angewendet [www.offis.uni-oldenburg.de/projekte/damm/KB_KORSYS.html]. Der hierbei verwendete Model-Checking-Ansatz verwendet ein Modell des Systems, das aus Statemate in Automaten übersetzt wird und die Darstellung der Eigenschaft in Formeln.

*Fazit:*

Wie die vorherigen Abschnitte gezeigt haben, besteht ein allgemein hohes Interesse an der Nutzbarmachung formaler Methoden für die Softwareentwicklung. Dies drückt sich durch ein breites Spektrum an universitären und industriellen Forschungs- und Entwicklungsaktivitäten aus und ist natürlich auch durch die zwingende Notwendigkeit einer Erhöhung der Softwaresicherheit begründet. Es wurde gezeigt, dass es sowohl Maßnahmen gibt, die im Vorfeld der Softwareentwicklung angewendet werden können, und dass es darüber hinaus Methoden und auch bereits Tools gibt, die eine Verbesserung der Softwarequalität während des Entstehungsprozesses erreichen. Weiterhin wurden einige Methoden und Verfahren vorgestellt, die auch nach der Erstellung der Software einsetzbar sind, um bestehende Fehler oder Unzulänglichkeiten zu beseitigen.

Dennoch bestehen allgemein große Befürchtungen, dass der sehr hohe Aufwand, der z. B. bei einer formalen Spezifikation betrieben werden muss, nicht durch die Ergebnisse gerechtfertigt wird [Lewe97]. Dieser Umstand kann nur durch Werkzeuge behoben werden, die schnell erlernbar und anwendbar sind und somit eine hohe Nutzerakzeptanz erwerben.

# 2 Anforderungen an ein formales Beschreibungsmittel in der Steuerungstechnik

Damit ein formales Beschreibungsmittel als geeignet für die Anforderungen der Steuerungstechnik gelten kann, muss es verschiedenen Bedingungen genügen, die sich aus den unterschiedlichen Betrachtungsweisen und Anforderungen der potentiellen Anwender ergeben. So ist es zwingend notwendig, die Funktionalität und vor allem auch die Benutzerschnittstellen einer zu entwickelnden Methode bzw. eines Tools auf die tatsächlichen Gegebenheiten und Erfordernisse des zukünftigen Einsatzes abzustimmen. Dieser Abschnitt beleuchtet einige dieser Aspekte und beschreibt dabei gleichzeitig wichtige Eigenschaften, die von einem formalen Beschreibungsmittel speziell im Bereich der Steuerungstechnik verlangt werden.

## 2.1 Allgemeine Anforderungen

Das Programmieren von Software, z. B. die Erstellung eines Steuerungsprogramms für eine Maschine oder Anlage ist ein kreativer Prozess, bei dem eine gestellte Aufgabe in ein korrektes Programm überführt werden muss. Dieser Prozess ist sehr komplex, so dass er auch in absehbarer Zukunft nur von Spezialisten zu bewältigen ist. Dabei stellt die Erstellung der Spezifikation das herausragende Problem dar. In [Litz99] wird dies so dargestellt: „Erst die Formalisierung, d. h. die Umsetzung der informellen in eine formale Spezifikation, ist der Schlüssel zur systematischen Lösung der Steuerungsaufgabe. Diese Umsetzung kann rechnerunterstützt, aber nicht automatisch erfolgen. Sie ist im Kern eine Leistung des Menschen ...“.

Die Erstellung eines Programms aus einer Idee heraus, lässt sich auch als Abbildung der Realität in ein Programm bezeichnen [Part98]. Diese Abbildung erfolgt in mehreren Schritten:

1. Aus der gegebenen Realität wird in einem „Erkenntnisprozess“ ein mentales Modell dieser Realität erzeugt.
2. Ein nachfolgender Abstraktionsprozess schafft ein mentales Modell des relevanten Ausschnittes der Realität bezüglich einer spezifischen Anwendung.
3. Der folgende Darstellungsprozess repräsentiert dieses mentale Modell (z. B. mit einem nichtformalen Beschreibungsmodell).
4. In einem Formalisierungsprozess erfolgt die formale Repräsentation (d. h. die Überführung in ein formales Modell).
5. Im Programmspezifikationsprozess werden die Spezifikationsdaten definiert.
6. Im abschließenden Implementierungsprozess erfolgt die Erstellung des Programms [Part98].

Die ersten fünf Schritte sind nichts anderes als der Spezifikationsprozess, den man selbst als eine Sequenz von Aktivitäten betrachten kann, die zur Entwicklung eines ersten Produkts, nämlich der Spezifikation, führt [Frap01]. Hierbei sind die Charakteristiken des Systems aufzustellen: seine funktionalen Anforderungen, Effizienzanforderungen oder auch Implementierungsanforderungen. Der Aufwand für die

anschließende Umsetzung der (hoffentlich korrekten) Spezifikation in das eigentliche Programm wird allgemein als vernachlässigbar klein beschrieben.

Um den Spezifikationsprozess sicherer und schneller zu gestalten und an die heutigen Bedürfnisse sich schnell ändernder Anforderungen anzupassen, müssen neue Methoden und Werkzeuge für den Softwareentwurf bereitgestellt werden. Einige der Anforderungen an zukünftige Methoden und Werkzeuge sind in [Litz99] dargestellt:

- sie müssen durchgängig in allen Phasen des Entwicklungsprozesses einsetzbar sein,
- sie müssen die Entwicklung hybrider technischer Systeme (kontinuierlich-diskret) sowie die Erstellung verteilter Realisierungen unterstützen,
- ein flexibler Simulationsteil muss den Test der Software bereits in frühen Phasen ermöglichen,
- die Software muss in genormten Darstellungen (z. B. nach DIN EN 61131-3) erstellt werden,
- eine automatische Codeerzeugung sowie die Analyse und Verifikation der Software soll möglich sein.

Die formale Spezifikation dient der eindeutigen Festlegung von Anforderungen an das Gesamtsystem [Lewe97]. Sie wird aus einer informellen Spezifikation, die zunächst nur gedanklich existiert, abgeleitet. Von der Überführung in die formale Spezifikation werden folgende Ergebnisse erwartet:

- die einengende Notation, die die Beschreibung der Anforderungen erschwert, soll zu einer größeren Eindeutigkeit zwingen,
- Inkonsistenzen der informellen Spezifikation sollen beim Formulierungsversuch explizit werden,
- Zwang zu einem einfacheren Entwurf, da nur von einfachen Entwürfen wichtige Eigenschaften formal bewiesen werden können,
- Fokussierung der Aufmerksamkeit der Entwerfer auf die durch die Formalismen ausgedrückten Systemaspekte,
- Eindeutigkeit durch präzise Semantik der benutzten Notationen,
- Konsistenzprüfung über mehrere Ebenen durch Überdeckung mehrerer Ebenen durch die Notation [Lewe97].

C. Aguilera et. al fassen diese Anforderungen an die Spezifikation folgendermaßen zusammen:

- sie muss sowohl für den Kunden als auch für den Entwickler verständlich sein,
- sie muss in allen Bereichen konsistent sein,
- jede einzelne Anforderung muss vollständig sein,
- jede Anforderung muss nachweisbar und auffindbar sein,
- die Implementierung muss testbar sein,
- es soll eine maximale Designfreiheit bei gleichzeitig minimaler Detaillierung bestehen [Agui90].

In [Alag98] sind weitere grundlegende Eigenschaften einer Spezifikation zusammengefasst worden:

- es muss möglich sein, das beobachtbare Verhalten zu definieren,
- das Interface einer Softwarekomponente muss exakt und einfach sein,
- das Verhalten des Ganzen muss ausdrückbar sein in Form des Verhaltens der Teile, bzw. es muss möglich sein, Spezifikationen zusammenzusetzen,
- es muss möglich sein, ein Programm aus einer detaillierten Designspezifikation zu entwickeln,
- die Designspezifikation muss eine Beschreibung des gesamten Verhaltens enthalten - ausgedrückt in einer Verhaltensspezifikation.

Als ein besonders wichtiges Kriterium wird in [Kohr91] und [Kohr93] eine abteilungsübergreifende Spezifikationsmethodik gefordert. Die systematische Erstellung von Steuerungsprogrammen erfordert präzise Vorgaben aus allen vorgelagerten und parallel arbeitenden Bereichen. Gerade mit dem Bereich der mechanischen Konstruktion gibt es enge Verzahnungen, wenn es darum geht, die Sensorik und Aktorik eines Systems zu spezifizieren oder zu verändern. Hieraus ergeben sich weitere Anforderungen:

- eine zentrale Informationshaltung muss Redundanzen vermeiden,
- existierende Unterlagen müssen wieder verwendbar und durchgängig durch alle Mitarbeiter nutzbar sein,
- die verwendeten Tools müssen von allen Mitarbeitern beherrscht und akzeptiert werden.

Beim Vergleich dieser Anforderungen mit den aktuellen Gegebenheiten wird deutlich, dass es gerade im Bereich der durchgängigen Anwendung von Spezifikationsmitteln große Defizite gibt. In jeder Branche und jedem Bereich haben sich spezielle eigene Vorgehensweisen herausgebildet, die einen Austausch von Informationen erschweren, wenn nicht gar verhindern.

Wie eingangs gezeigt, stellt der Vorgang des Programmierens einen kreativen Prozess dar, der eine Überführung geforderter Abläufe und Aktionen in maschinenausführbare Algorithmen beinhaltet. Hierbei stellt sich zunächst die Frage, wie diese Anforderungen und Erwartungen an das System ermittelt, geordnet und derart abgelegt werden, so dass sie auch später wieder verwendet werden können. Die Präsentation der dahinter liegenden Informationen und deren Nutzbarmachung als konkretes Wissen um den Prozess und das System erfordern einen Formalismus, der die Akquisition des erforderlichen Wissens aus einer Quelle in einer Form ermöglicht, die dieses Wissen sowohl für den Menschen als auch für die Maschine verständlich repräsentiert und zur möglichst effizienten Problemlösung beiträgt (siehe [Bibe93]). Der Autor führt weiter an, dass mit der Formalisierung des Wissens, der Formalisierung des Problems und einer geeigneten Verarbeitung unmittelbar die Lösung des Problems gegeben ist.

Es gibt die unterschiedlichsten Formalismen zur Repräsentation des Wissens [Bibe93]:

- natürliche Sprache in Form geschriebenen oder gesprochenen Textes,
- Bilder, Grafiken,
- Fregesche Repräsentationen (Prädikatenlogik, Formeln, die eine äquivalente Darstellung natürlichsprachlicher Sachverhalte enthalten),
- assoziative Netze,
- Repräsentationssprachen, wie KL-ONE,
- neuronale Netze.

Bei der Überführung von der natürlichen Umgangssprache zur formalen Sprache wird immer ein Bruch entstehen. Dieser ist unvermeidlich, da ein semantisch mehrdeutiges Problem nicht aufgrund logischer Schlussweisen in ein eindeutiges überführt werden kann. Die Spezifikation führt prinzipiell zur Lösung der Aufgabe. Diese Lösung ist zwar noch nicht in der Basissprache der Maschine formuliert, somit ist noch keine Entscheidung für die Implementierung gefällt.

## 2.2 Formale Anforderungen

Ein zu entwickelndes formales Beschreibungsmittel, das Anforderungen an ein Steuerungssystem mit Hilfe der natürlichen Sprache beschreiben soll, ist eine künstliche Sprache. Künstliche Sprachen besitzen ebenso wie natürliche Sprachen eine Syntax, Semantik und Grammatik. Im Gegensatz zur natürlichen Sprache ist ihre Semantik aber eindeutig [Reck91]. Es ist notwendig, sämtliche Mehrdeutigkeiten und Möglichkeiten für missverständliche Formulierungen zu eliminieren. Aus diesem Grund werden künstliche Sprachen formal definiert, als Sprachbeschreibungssprache dient eine Metasprache, die Backus-Naur-Form (BNF).

Die BNF ist ein Formalismus zur Beschreibung kontextfreier Grammatiken und ist somit auch zur Syntaxdefinition geeignet. Grundlage bildet ein Alphabet, dessen Bestandteile als Terminalsymbole bezeichnet werden. Aus diesen Terminalsymbolen lassen sich unter Verwendung von Produktionsregeln Nichtterminalsymbole (Zeichenfolge von Terminal und Nichtterminalsymbolen) ableiten [Krau00].

Der GMA-Fachausschuss 7.21 „Formale Prozessbeschreibungen“ hat in seinem Richtlinienentwurf (Stand 2001), zur VDI/VDE-Richtlinie „Mensch-Prozess-Kommunikation unter besonderer Berücksichtigung der formalen Prozessbeschreibungen“ den Begriff der formalen Prozessbeschreibung folgendermaßen definiert. Notwendig sind:

- eine definierte Semiotik (die Menge von Symbolen und Zeichen, die die Sprache bestimmen),
- eine definierte Syntax (die Menge von Regeln für zulässige Zeichenkombinationen und Operationen bzw. Kombinationen),
- und eine formale Semantik (die definierte Menge von Operationen mit Symbolen und Symbolkombination nach Maßgabe der Zeichenbedeutung) [Ahre00].

Nach [Alag98] müssen Spezifikationssprachen die folgenden Charakteristiken enthalten:

- Formalismus: definierte Syntax und formale Semantik,
- Abstraktion: es gibt Primitive für die Definition und Manipulation von Daten auf dem logischen Level,
- Modularität: Konstruktionen zur Erweiterung einer Spezifikation durch „Anreicherung“ und „Komposition“.

In diesem Zusammenhang wird auch auf die Arbeit des GMA-Unterausschluss 1.8.1 hingewiesen, der u. a. für diese Beschreibungsmittel qualitative Kriterien zusammengestellt hat:

- Formale Basis,
- Möglichkeiten zur Abbildung der Systemstruktur, Modularität, Hierarchie und Dynamik,
- Kausalität (Logik von Abläufen), Abbildung paralleler Prozesse, Abbildung sequentieller Prozesse,
- Abbildung der Zeit,
- Simulierbarkeit (zusammen mit einem Tool), Analysierbarkeit, Diagnose,
- Phasenorientierte Bewertung: durchgängige Anwendung für Spezifikation, Modellbildung, Implementierung und Betrieb.

## 2.3 Anforderungen aus dem Bereich der Steuerungstechnik

Ein weiteres wichtiges Kriterium einer Spezifikationsmethodik ist ihr potenzielles Einsatzgebiet. Bei der in dieser Arbeit vorgestellten Sicherheitsfachsprache ist dies die Spezifikation von Steuerungsprogrammen für industrielle Fertigungsanlagen. Deshalb muss zunächst definiert werden, welche Strukturen in solchen Fertigungsanlagen bereits gegeben sind.

### *2.3.1 Strukturen von automatisierten Systemen*

Moderne Fertigungsbetriebe sind typischerweise hierarchisch in verschiedene Betriebsebenen unterteilt. Wenn auch die in der Literatur dargestellten Gliederungen selten exakt übereinstimmen (so wie auch ein Betrieb selten exakt wie ein anderer aufgebaut ist), so lässt sich dennoch eine gemeinsame Struktur herausarbeiten, (Abbildung 6, vergleiche [Polk92], [Kasp94], [Beus94], [Ahre00]).

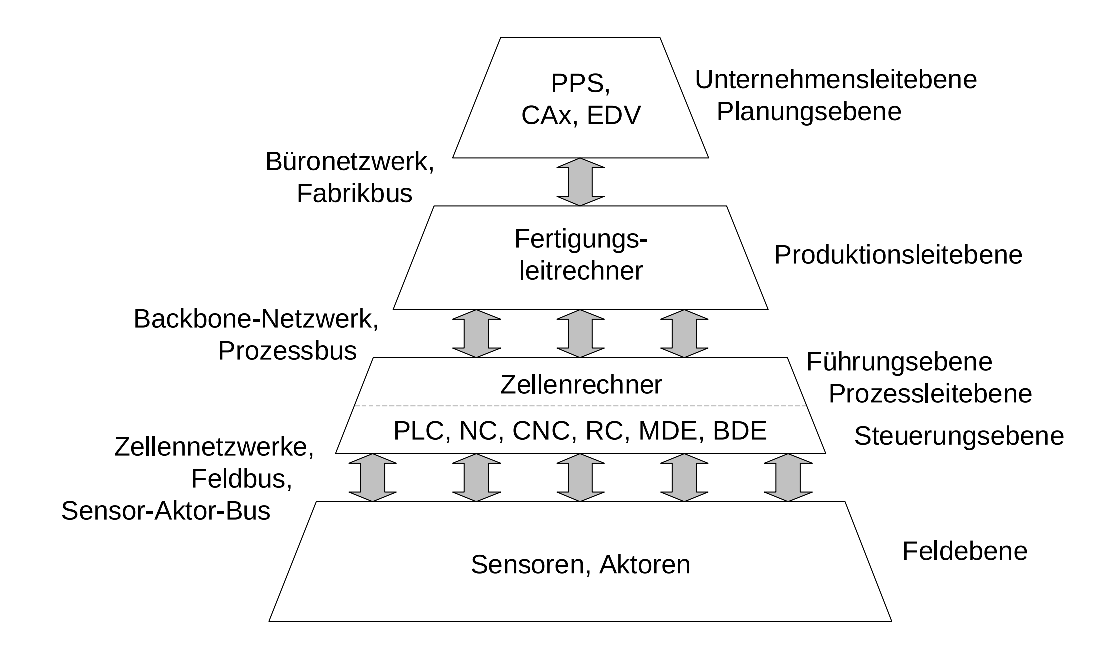



*Abbildung 6 - Betriebliches Ebenenmodell*

Man findet eine hierarchische Gliederung des betrieblichen Ebenenmodells in vier oder auch fünf Ebenen, wobei die Führungs- und die Steuerungsebene immer weiter miteinander verschmelzen. Insgesamt lässt sich feststellen, dass die bisherigen Strukturen durch verstärkten ebenenübergreifenden Einsatz von Netzwerken und Bussystemen immer weiter reduziert werden. Moderne Systeme ermöglichen sogar die Durchgängigkeit eines einzelnen Bussystems (z. B. Ethernet) durch den gesamten Betrieb.

Bezogen auf ihre Arbeitsinhalte und Aufgaben lassen sich zwischen diesen Ebenen große Unterschiede feststellen: während in den oberen Ebenen (den Leitebenen) dispositive Aufgaben im Vordergrund stehen, gehen diese in den unteren Ebenen zu operativen Aufgaben über. Bei der Analyse der Aufgabenstellungen und der dabei verwendeten Informationen werden zwei grundsätzliche Fakten deutlich: von oben

nach unten erfolgt eine Präzisierung und Verfeinerung der Aufgaben, während in der Gegenrichtung eine zielgerichtete Verdichtung der zugehörigen Informationen stattfindet.

In der *Feldebene* befinden sich die Mess- und Stelleinrichtungen. Diese liefern Prozesssignale (Primärdaten) zur Verarbeitung an die Prozessleit-/ Steuerungsebene, und erhält von dieser Stellbefehle zum Eingriff in den Prozess. In der Feldebene finden die Erfassung und Beeinflussung der Prozessgrößen und eine Signalaufbereitung statt.

Die *Prozessleit- bzw. Steuerungsebene* hat bereits leittechnische Funktionen, sie dient zum Führen von einzelnen Verfahrensgruppen oder Apparaten. Hier erfolgen die Umsetzung der Produktionsaufträge in verfahrenstechnische Realisierungsprozesse sowie die Koordinierung der dafür notwendigen Geräte. Die technische Ausstattung dieser Ebene besteht in Prozessdatenverarbeitungsgeräten, SPS, Robotersteuerungen, NC, CNC sowie RC und weiteren Geräten zur Betriebsdatenerfassung. Funktionen wie Steuern und Regeln, die Gewährleistung der Sicherheit und Störungsüberwachung, das An- und Abfahren oder die Rezeptursteuerung sind hier angesiedelt.

Die *Produktionsleitebene* dient zum Führen der Fabrik. Produktionsvorgaben werden hier übernommen und detailliert. Es werden Kapazitäten geplant und Aufträge abgewickelt, weitere Funktionen sind die Bestandsdisposition, Lagerverwaltung, Qualitätskontrolle, Logistik und Optimierung.

Die *Unternehmensleitebene* betrifft vor allem betriebswirtschaftliche Bereiche, also die Führung des gesamten Unternehmens. Sie ist die zielsetzende langfristig planende Ebene, sie beurteilt den Abnehmermarkt, steuert den Einkauf und erstellt Produktionsvorgaben.

Ein allgemeingültiges formales Beschreibungsmittel muss die Darstellung von Anforderungen in allen betrieblichen Ebenen ermöglichen. Dies betrifft inhaltlich hauptsächlich die Beschreibung eines geforderten Ablaufes, muss aber auch verschiedene zeitliche Horizonte (wie dispositive Anforderungen in den oberen Hierarchieebenen) berücksichtigen.

### *2.3.2 Berücksichtigung der Arbeitsweise einer Steuerung*

An dieser Stelle erfolgt eine kurze Einführung zur Arbeitsweise einer speicherprogrammierbaren Steuerung, da sich hieraus weitere grundlegende Schlussfolgerungen bezüglich der Formulierung von Anforderungen an ein Steuerungsprogramm und somit auch an den Aufbau der Sicherheitsfachsprache ergeben.

Die folgende Abbildung zeigt den allgemeinen Aufbau einer automatisierten Anlage (vergleiche [Lemm95]).

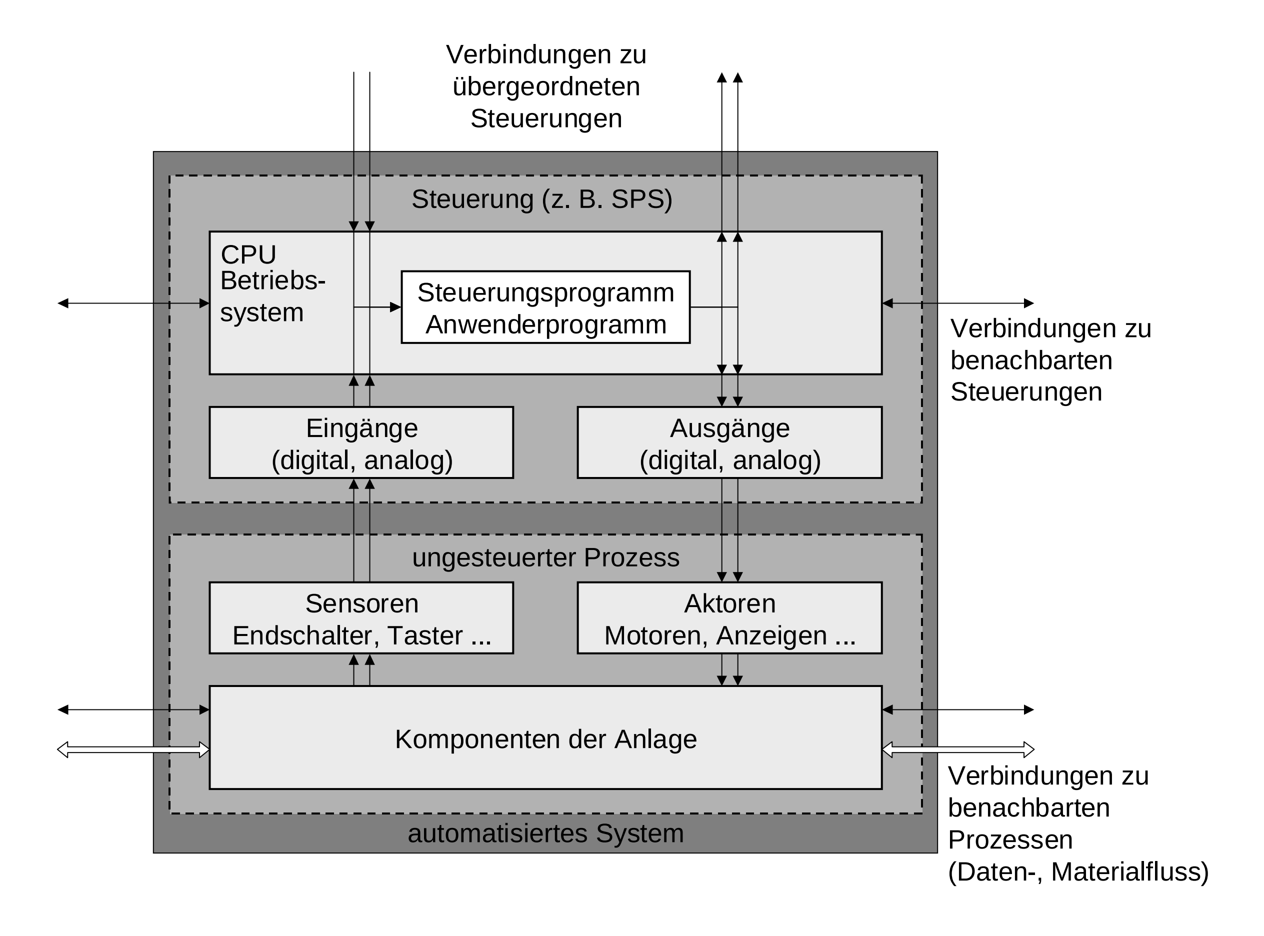



*Abbildung 7 - Informationsfluss in einem automatisierten System*

Eine SPS arbeitet immer mit einer Maschine oder Anlage zusammen, sie empfängt Informationen aus der Umgebung, verarbeitet diese entsprechend einem vorgegebenen Algorithmus und gibt wiederum Informationen an die Umgebung ab. Die Informationen der Umgebung werden durch Sensoren aufgenommen und in elektrische Signale umgewandelt. Diese werden über digitale oder analoge Eingangsbaugruppen in das SPS-System eingebracht und durch das Betriebssystem eingelesen. Diese Eingangssignale werden entsprechend dem Steuerungsprogramm verarbeitet. Die berechneten Ergebnisse werden wiederum durch das Betriebssystem an die Ausgangsbaugruppen geliefert. Diese übermitteln die neuen Anweisungen als elektrische Signale an die Aktoren der Maschine, wo die gewünschten Aktionen ausgeführt werden.

Eine speicherprogrammierbare Steuerung zeichnet sich durch eine zyklische Arbeitsweise aus. Der zuvor beschriebene Prozess erfolgt in einem festgelegten und konstanten Rhythmus, der auch als *SPS-Zyklus* bezeichnet wird. Dabei werden zu Beginn des Zyklusses die Signale der Umgebung eingelesen und zwischengespeichert. Mit diesen Werten und internen Variablen des Steuerungsprogramms werden danach die Berechnungen durchgeführt. Nach Beendigung des Zyklus werden die Ergebnisse der Berechnungen an die Umgebung ausgegeben. Zusätzlich gibt es noch eine gewisse Wartezeit, bis der SPS-Zyklus wieder von vorn gestartet wird. Der

Neustart des SPS-Zyklusses erfolgt in einem konstanten Zeitraster, der *Zyklus-* oder *Abtastzeit*.

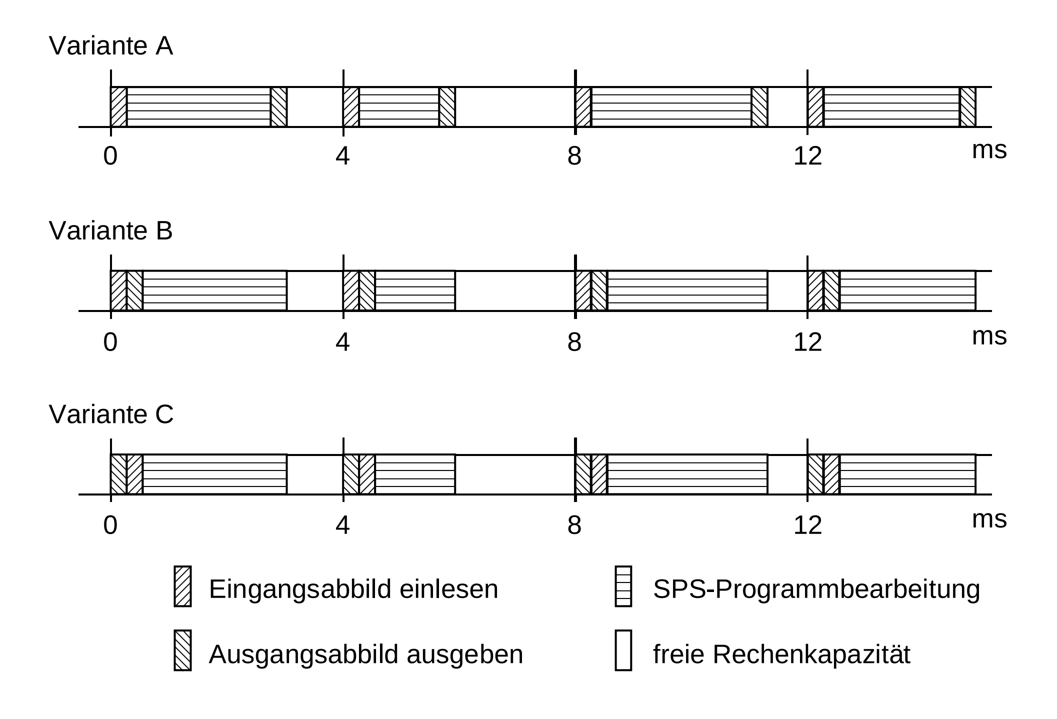



*Abbildung 8 - Zyklische Arbeitsweise einer SPS*

Die Reihenfolge der dargestellten Teilprozesse wird in der Literatur recht unterschiedlich beschrieben (Varianten A, B, C). Allen Varianten ist gemeinsam, dass zwischen dem Eintreten bestimmter Ereignisse in der Umgebung und der entsprechenden Reaktion durch die SPS immer eine gewisse Zeit vergeht. Im ungünstigsten Fall tritt ein Ereignis genau dann ein, wenn die SPS gerade begonnen hat, das Programm zu bearbeiten. Das Eingangssignal kann also erst mit dem darauf folgenden Zyklus eingelesen und verarbeitet werden. Die worst-case-Reaktionszeit einer SPS beträgt demnach das Doppelte der Zykluszeit, im günstigsten Fall beträgt sie immerhin auch noch die einfache Zykluszeit. Diesen SPS-typischen Umständen muss bei der späteren Formulierung der Anforderungen unbedingt Rechnung getragen werden.

Die Berücksichtigung des SPS-Zyklusses in Rahmen der Sicherheitsfachsprache erfolgt durch spezielle Zeitangaben bei der Formulierung der Anforderungen sowie durch Einführung von Monitoring-Variablen in den temporallogischen Formeln (siehe Abschnitte 3.1.4 bzw. 3.2.4).

## 2.4 Inhaltliche Anforderungen

Die Spezifikation, also die technologischen und technischen Vorgaben für die gewünschte Funktionsweise einer Maschine oder einer Software, wird üblicherweise in Kooperation zwischen Auftraggeber und Auftragnehmer erstellt und in einem Pflichtenheft schriftlich festgehalten. Die Darstellung erfolgt gewöhnlich zunächst als verbale Beschreibung der Anforderungen oder mit anderen spezifischen Darstellungsformen. Nach [Somm97] können die Systemanforderungen in funktionale und nichtfunktionale Anforderungen unterteilt werden. Funktionale Anforderungen beschrei-

ben, was das System tun soll (manchmal auch, was es nicht tun soll). Die nichtfunktionalen Anforderungen beschreiben weitere Bedingungen, die erfüllt werden müssen, wie z. B. die Gestaltung des Entwicklungsprozesses, das Einhalten von Normen usw. Hierdurch bestimmen sich auch die Validierungskriterien des Systems: zum einen die Eigenschaften, die vom System erwartet werden (fehlerfreies Verhalten) und zum anderen nicht erwünschte Eigenschaften, deren Abwesenheit zu überprüfen ist [Flei97].

### *2.4.1 Das Pflichtenheft*

Bei der Festlegung der gewünschten Eigenschaften eines Produktes unterscheidet man zwischen der Anfertigung eines Lastenheftes und eines Pflichtenheftes. Das Lastenheft ist vom Auftraggeber vollständig und widerspruchsfrei zu erstellen. Es enthält eine Zusammenstellung aller Anforderungen des Auftraggebers hinsichtlich Liefer- und Leistungsumfang, also das „Was“ und „Wofür“. Das Pflichtenheft wird vom Auftragnehmer unter Beachtung der im Lastenheft genannten Anforderungen erstellt. Es enthält die Beschreibung der Realisierung aller Anforderungen des Lastenheftes, also das „Wie“ und „Womit“. Das Pflichtenheft zu einem Steuerungsprogramm muss alle Informationen enthalten, die für die vollständige, eindeutige und korrekte Beschreibung aller Funktionen des Programms notwendig sind. Die [VDI3694] legt den Aufbau eines Pflichtenheftes fest (siehe auch [Balz96]).

Zu den Bestandteilen eines Pflichtenheftes gehört neben einer verbalen Kurzbeschreibung des Programms die Darstellung der verfügbaren Input-Parameter (Sensoren, Bedienereingaben) und die Beschreibung der notwendigen Output-Parameter (Aktoren, Anzeigen, Ausgaben usw.). Die Funktionsbeschreibung soll dabei allgemeinverständlich, aber auch übersichtlich sein. Zu diesem Zweck haben sich in der Vergangenheit verschiedene grafische Darstellungsformen, wie Funktionsdiagramme [VDI3260], Funktionspläne [DIN 40719/6] und auch Petri-Netze, als praktikabel erwiesen.

Das Pflichtenheft umfasst primär die gewünschten *Funktionsanforderungen* eines Systems, d. h. die Funktionen, die während des normalen, standardmäßigen Betriebs des Systems auftreten (*Betriebsfunktionen*). Darüber hinaus zählen zu den Funktionsanforderungen aber auch die Reaktionen des Programms auf ungewöhnliche, außerplanmäßige Ereignisse, wie Störungen oder Not-Aus (*Störungsfunktionen*). Während der Normalbetrieb eines Systems noch relativ einfach beschrieben werden kann, müssen für die Beschreibung der Programmreaktionen auf anormale Ereignisse weitaus mehr globale Systemzusammenhänge betrachtet werden. Betriebsfunktionen werden während des Normalbetriebs ständig durchlaufen, Störungsfunktionen werden nur im Ausnahmefall aktiviert, für sie ist ein Test oder eine Prüfung in der realen Anlage meist nicht mehr möglich. Eine gemeinsame Eigenschaft der Funktionsanforderungen besteht darin, dass bestimmte Ereignisse und Zusammenhänge eine eindeutige Reaktion innerhalb einer fest definierten und hinreichend minimalen Reaktionszeit verlangen (z. B. Notabschaltungen).

Das Pflichtenheft beinhaltet außer den notwendigen Abläufen des Programms aber auch Festlegungen, welche Programmreaktionen unbedingt vermieden werden müssen. Diese *Sicherheitsanforderungen* lassen sich jedoch meist nicht so einfach wie die bereits erwähnten Funktionseigenschaften beschreiben. Es ist oft problematisch,

die kritischen Situationen eines Steuerungssystems vollständig zu identifizieren, bzw. die komplexen Einflussfaktoren und Wechselwirkungen der Systeme untereinander zu erkennen.

Das maßgebliche Problem besteht jedoch darin, dass es keine ingenieurtechnische Darstellungsform für die Darstellung auszuschließender Reaktionen eines Steuerungssystems gibt.

### *2.4.2 Kategorisierung aufgrund des Analyseverfahrens*

Bei einer Verifikation durch Model-Checking werden die Anforderungen an ein System an einem Modell des Systems überprüft. Die Untersuchung erfolgt am Erreichbarkeitsgrafen, einer Repräsentation aller erreichbaren Zustände des Systems. Durch diese Vorgehensweise ergeben sich verschiedene Klassen analysierbarer Eigenschaften. Man kann zunächst eine Unterteilung bezüglich der zu untersuchenden Systemzustände vornehmen, die im Folgenden als ***statische*** bzw. ***dynamische*** Analysen bezeichnet werden.

Statische Analysen untersuchen entweder:

- die prinzipielle Erreichbarkeit eines erwünschten Zustandes, sie zeigen mit einem Zeugniszustand, dass das System diesen Zustand überhaupt einnehmen kann (ohne den Weg dorthin zu betrachten),
- die prinzipielle Abwesenheit eines (möglicherweise unerwünschten) Zustandes, dies betrifft die typischen Sicherheitsanforderungen an ein System.

Dynamische Analysen untersuchen:

- die prinzipielle Erreichbarkeit eines erwünschten Zustandes von einem gegebenen Anfangspunkt aus, hierdurch kann nachgewiesen werden, dass das System einen definierten Zielzustand (dessen Existenz durch eine statische Analyse bereits nachgewiesen wurde) auch von einem oder mehreren definierten Startpunkten erreicht werden kann,
- die schnellstmögliche Erreichbarkeit eines erwünschten Zustandes durch Identifizierung des kürzesten Pfades (unter Beachtung der Programmausführung),

In [Wehr96], [Mann92] und anderen Quellen werden relevante Kategorien von Eigenschaften definiert. Man unterscheidet demnach:

- Sicherheitseigenschaften,
- Fortschrittseigenschaften,
- Lebendigkeitseigenschaften.

Die Klassifizierung ist in der Literatur leider nicht durchgängig, gegebenenfalls tritt eine Vermischung der Fortschritts- und Lebendigkeitseigenschaften auf.

Sicherheitseigenschaften (engl. safety properties) beinhalten, dass das System niemals in einen unerwünschten Zustand gerät. Lebendigkeitseigenschaften (engl. liveness properties) drücken aus, dass das System irgendwann in einen bestimmten er-

wünschten Zustand geraten kann. Fortschrittseigenschaften verschärfen dies dadurch, dass das System auch in diesen geforderten Zustand kommen muss.

### *2.4.3 Anforderungen aus dem Bereich der Steuerungstechnik*

Ausgehend vom vorherigen Abschnitt wird nun gezeigt, welche speziellen Anforderungen an ein Steuerungsprogramm gestellt werden und wie diese kategorisiert werden können.

Die festgelegten Vorgaben an das Steuerungsprogramm werden im Folgenden allgemein als ***Anforderungen*** (engl. *requirements*) bezeichnet. Im vorherigen Abschnitt wurde bereits eine grobe Einteilung dieser Anforderungen in Funktions- und Sicherheitsanforderungen dargelegt. Bei der Formulierung dieser Anforderungen werden bestimmte Zustände des Systems betrachtet, es werden bestimmte Aussagen über diese Systemzustände getroffen sowie spezielle Zusammenhänge zwischen verschiedenen Systemzuständen hergestellt.

Funktionsanforderungen zeichnen sich dadurch aus, dass sie zunächst einen bestimmten Ausgangszustand beschreiben, von dem aus ein anderer Systemzustand (Zielzustand) erreicht werden muss. Der Ausgangs- und der Zielzustand unterscheiden sich dabei mindestens um den Wert einer einzelnen Systemvariablen (z. B. die Aktivität eines Motors). Ein typisches Beispiel für eine solche Funktionsanforderung ist, dass bei Betätigung eines bestimmten Tasters durch den Anlagenbediener unverzüglich ein Motor einzuschalten ist.

Diese genannten Anforderungen werden weiterhin als ***Forderungen*** (engl. *demands*) bezeichnet. Sie beschreiben die notwendigen und erforderlichen Reaktionen des Steuerungsprogramms auf festgelegte Ereignisse.

Forderungen lassen sich weiterhin hinsichtlich ihres zeitlichen Horizonts betrachten. Man kann auf diese Weise kurzfristige und langfristige Programmanforderungen beschreiben. Unter kurzfristigen Anforderungen kann man solche verstehen, die eine möglichst schnelle Reaktion des Steuerungsprogramms auf das Eintreten bestimmter Ereignisse beschreiben. Ein Beispiel hierfür ist das notwendige Abschalten eines Motors, sobald ein durch ihn bewegtes Maschinenelement einen bestimmten Endlagenschalter erreicht hat und somit ein Signal ausgelöst wurde. Bei langfristigen Anforderungen ist der Zeitpunkt für das Erreichen der beschriebenen Reaktion nicht exakt definiert. Hierbei werden übergeordnete Systemziele definiert, so z. B., dass ein Werkstück, das in eine Produktionszelle gelangt, diese auch irgendwann wieder verlassen muss.

Bei Sicherheitsanforderungen werden üblicherweise einzelne Zustände des Systems beschrieben, die innerhalb der Betrachtungszeit des Systems niemals erreicht werden dürfen. Beispiele hierfür sind Zustände, die zu einer Gefährdung von Personen, der Umgebung, von Material oder der Maschine selbst führen können. So könnte für eine betrachtete Maschine festgelegt sein, dass es niemals eine Situation geben darf, in der eine Sicherheitsvorrichtung geöffnet ist und gleichzeitig eine Maschinenbewegung ausgeführt wird.

Sicherheitsanforderungen werden im Folgenden als ***Verbote*** (engl. *prohibitions*) bezeichnet. Sie beschreiben Zustände des Steuerungsprogramms, die nicht eintreten dürfen, bzw. verbotene Reaktionen auf festgelegte Ereignisse.

Es erweist sich als brauchbar und während der Software-Entwicklungsphase als durchaus üblich, eine weitere Kategorie von Anforderungen zu definieren. So gibt es Situationen, in denen eine bestimmte Reaktion des Steuerungsprogramms zwar erlaubt ist, aber noch nicht ausgeführt werden darf, weil eine andere, zusätzliche Bedingung noch nicht erfüllt ist. Eine solche typische Freigabesituation ist gegeben, wenn z. B. eine Maschinenbewegung ausgeführt werden darf, wenn eine Sicherheitsvorrichtung geschlossen ist, die Aktion aber erst dann erfolgen soll, wenn gleichzeitig auch ein bestimmter Taster betätigt wurde.

Diese Anforderungen werden im Folgenden als ***Möglichkeiten*** (engl. *possibilities*) bezeichnet. Sie beschreiben erlaubte Reaktionen des Steuerungsprogramms auf festgelegte Ereignisse.

## 2.5 Einbindung der Sicherheitsfachsprache in eine Verifikationsumgebung

Die Sicherheitsfachsprache wurde nicht losgelöst in einem Einzelprojekt erarbeitet, vielmehr ist sie Bestandteil eines Forschungsprojektes ([Hein97a], [Meie98], [Hein00b]), das sich mit der Entwicklung von Methoden und Werkzeugen zur Zertifizierung von SPS-Anwenderprogrammen beschäftigt. Ziel dieses Forschungsprojektes ist es, bekannte Methoden und Verfahren zur Programmverifikation aus der Informatik auf den speziellen Bereich der Steuerungstechnik zu adaptieren.

Im erwähnten Forschungsprojekt wurde speziell das Model-Checking-Verfahren ausgewählt, dieses gliedert sich, ebenso wie das Gesamtprojekt, in zwei Hauptbereiche (Abbildung 9). Die notwendigen Grundlagen für eine Verifikation durch Model-Checking sind das Vorhandensein eines Modells des zu analysierenden Systems (linker Bereich der Abbildung) sowie eine formale Spezifikation der Funktions- und Sicherheitseigenschaften, die durch dieses System erfüllt werden sollen (rechter Bereich der Abbildung). Das zu untersuchende System besteht aus einer speicherprogrammierbaren Steuerung, dem Steuerungsprogramm und der zu steuernden Anlage. Das eigentliche Untersuchungsobjekt ist jedoch nur das Steuerungsprogramm (auch SPS-Programm, Anwenderprogramm), da die Anlage und die verwendete Steuerung als feste, unveränderliche Komponenten angenommen werden müssen.

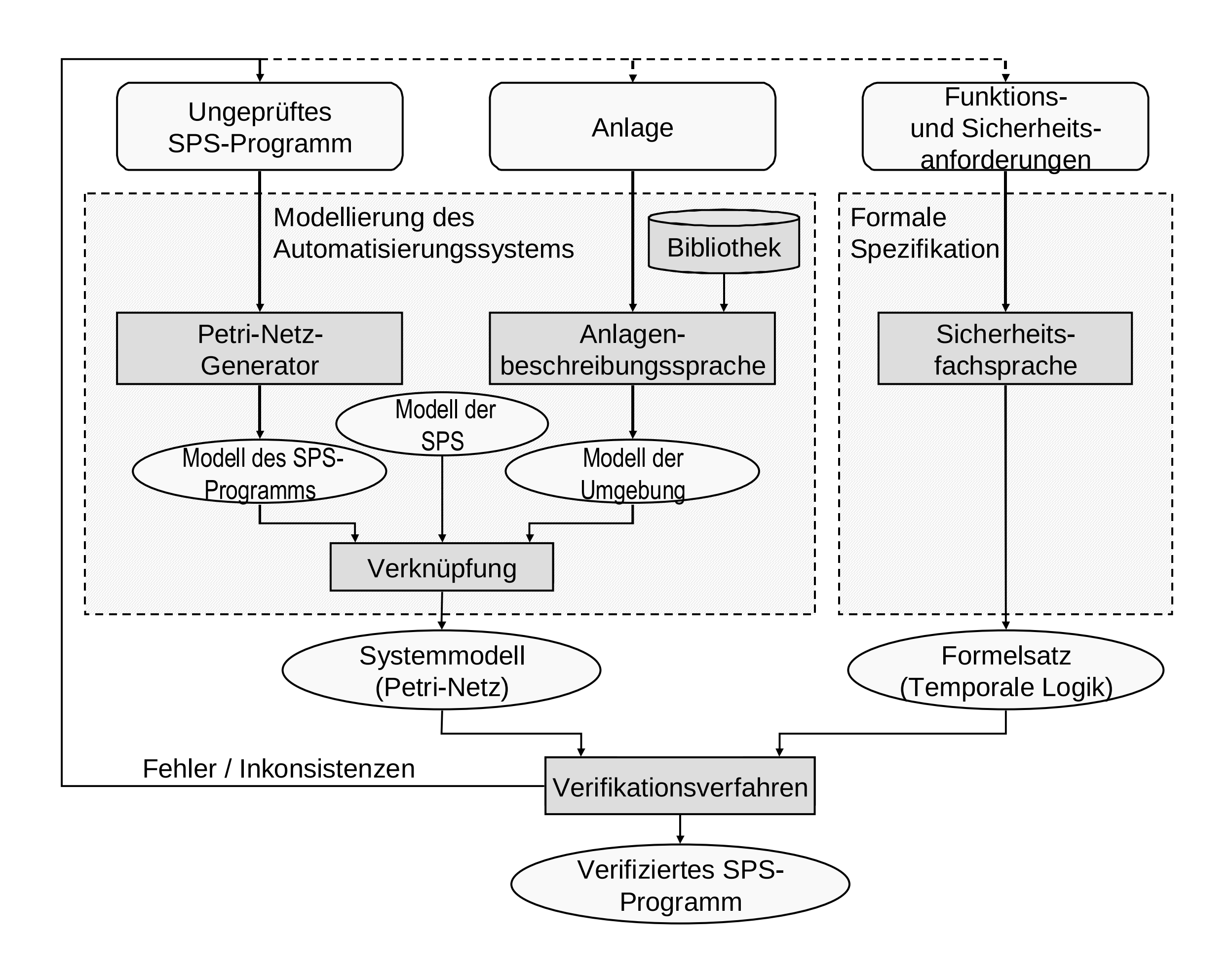



*Abbildung 9 - Übersicht über das Verifikationsverfahren*

Das SPS-Programm wurde durch einen Automatisierungstechniker mit herkömmlichen Methoden entworfen und programmiert, es wird dann auf der SPS ausgeführt. Die zu steuernde Anlage ist entweder direkt oder über ein Bussystem mit der SPS verbunden. Hierfür gibt es einen Beschaltungsplan, der eine Auflistung der verwendeten Aktoren und Sensoren, sowie spezifische Details der Signalparameter enthält. Dieser Beschaltungsplan ist die Grundlage für die Verbindung der Aktoren und Sensoren mit den Ausgangs- und Eingangsbaugruppen der SPS. Gleichzeitig dient er auch als inhaltliche Schnittstelle zwischen dem SPS-Programm und der Anlage, da er auch eine Auflistung der Signale darstellt, die im SPS-Programm wieder zu benutzen sind. Die Schaffung des Modells des Automatisierungssystems, das als Basis für den Verifizierungsvorgang dienen soll, umfasst somit drei Komponenten:

1. die zu steuernde Anlage mit ihren Sensoren und Aktoren,
2. die verwendete Steuerung mit den integrierten Betriebssystemfunktionen,
3. das vom Automatisierungstechniker erstellte Steuerungsprogramm.

Diese drei Komponenten werden einzeln modelliert und in einem anschließenden Kompositionsschritt zu einem Systemmodell vereinigt.

Die am Modell zu verifizierende Spezifikation ist durch die Anforderungen des Pflichtenheftes gegeben. Diese Festlegungen liegen jedoch in automatisierungstechnisch spezifischen Darstellungsformen bzw. verbal vor. Da für das Model-Checking

jedoch Formeln der Temporalen Logik benötigt werden, müssen die Anforderungen zunächst formalisiert und in die benötigten Formeln überführt werden. Die hierbei verwendete Sicherheitsfachsprache und die notwendigen Schritte werden in den Kapiteln 3.1 und 4.2 dargestellt.

Die anschließende Überprüfung der Anforderungen erfolgt im Verifizierungsschritt durch den Model-Checker. Sollten alle Anforderungen erfüllt und als richtig verifiziert worden sein, kann das Steuerungsprogramm als korrekt bezüglich den gegebenen Bedingungen bezeichnet werden. Treten dagegen Fehler auf, ist eine Modifikation des SPS-Programms notwendig.

# 3 Die Sicherheitsfachsprache

In den vorherigen Abschnitten wurden verschiedene Anforderungen an ein formales Spezifikationswerkzeug dargestellt, das den spezifischen Gegebenheiten eines Einsatzes in der Steuerungstechnik gerecht werden soll. Ausgehend von diesen Anforderungen wurde die Sicherheitsfachsprache entwickelt, (siehe [Meie99], [Hein01] und [Mert01]), die folgende Anforderungen erfüllen soll:

1. eindeutige natürlichsprachliche Formulierung von steuerungstechnischen Anforderungen,
2. durchgängige und bereichsübergreifende Anwendung in allen Phasen der Softwareentwicklung,
3. Abdeckung aller steuerungstechnisch relevanten Kategorien, auch solcher, die mit herkömmlichen Methoden nicht darstellbar sind.

## 3.1 Definition der SFS

Herausragende Anforderung an die Sicherheitsfachsprache ist die Verwendung der natürlichen deutschen Sprache für die Spezifikation. Deshalb ist es notwendig, sich mit dem Aufbau und den Strukturen der deutschen Sprache auseinander zu setzen. Die natürliche Sprache, mit der wir täglich umgehen, unterliegt bestimmten Regeln ([Flie86], [Kürs93]). Sie hat, wie eine künstliche Sprache, eine Semiotik, Syntax und Grammatik. Ihre Semantik ist jedoch nicht eindeutig, die bestehenden Mehrdeutigkeiten wurden bereits erwähnt.

Ziel ist es deshalb nicht, ein komplexes System zur Sprach- bzw. Texterkennung zu entwickeln, wofür es andere vielversprechende Forschungsprojekte gibt. Vielmehr wird mit der Sicherheitsfachsprache ein Weg eingeschlagen, der eine stärkere Reglementierung der formulierbaren Sätze benutzt. Vergleichbare deutschsprachige Ansätze hierzu sind: [Hölz98a] – Nutzung von „Schablonen“, [Bits02] – Nutzung von „Safety Patterns“. Innerhalb der Sicherheitsfachsprache wurden 18 Kategorien von Anforderungen definiert, die sich sprachlich und in ihrer Bedeutung vollständig unterscheiden. Wie diese Kategorien ermittelt und dargestellt wurden, wird in den folgenden Abschnitten dargestellt.

### *3.1.1 Ausdrucksmöglichkeiten der natürlichen deutschen Sprache*

Jede Anforderung, die mit der Sicherheitsfachsprache formuliert wird, ist eine Regel. Regeln sind ein weitverbreitetes Mittel zur Wissensrepräsentation [Balz96]. Sie bestehen aus einer Vorbedingung (*„wenn“*) und einer Aktion (*„dann“*). Man kann zwei Typen von Regeln unterscheiden:

- Implikationen / Deduktionen, mit denen der Wahrheitsgehalt einer Feststellung hergeleitet wird (*„wenn ... dann ist ...“*),
- Handlungen, mit denen ein Zustand verändert wird: (*„wenn ... dann wird ...“*).

Die im Abschnitt 2.4.3 definierten Anforderungskategorien zeichnen sich in ihrer Anwendung bei der verbalen Formulierung der Programmanforderung durch die Verwendung bestimmter sprachlicher Konstruktionen und typischer Schlüsselwörter aus.

| **Kategorie** | **Beispiele** |
|---|---|
| Forderungen | *muss, müssen, soll, sollen, ist zu ..., sind zu ...* |
| Verbote | *darf nicht, soll nicht* |
| Möglichkeiten | *darf, dürfen, kann, können* |

*Tabelle 1 - Schlüsselwörter zur Formulierung der Anforderungskategorien*

Sprachtechnisch gesehen wird der Inhalt einer Anforderung durch diese Modalverben ausgedrückt [Flie86].

Weiterhin lassen sich bei der Analyse der verbalen Formulierungen Schlüsselwörter identifizieren, die bestimmte zeitliche, logische oder inhaltliche Zusammenhänge zwischen verschiedenen Aussagen herstellen.

| **Kategorie** | **Beispiele** |
|---|---|
| Zeitliche Zusammenhänge | *nach, nachdem, danach, vor, bevor, zuerst, sofort, dabei, bis, solange, nie, niemals* |
| Logische Zusammenhänge | *und, oder, nicht* |
| Bedingungen, Konditionale, Implikationen | *wenn ... dann ..., falls ...* |
| Erweiterte Bedingungen, Bikonditionale, Äquivalenz | *nur dann wenn ..., genau dann wenn ...* |

*Tabelle 2 - Auflistung weiterer Schlüsselwörter*

Aus sprachtechnischer Sicht handelt es sich hierbei um Konjunktionen, die Sätze oder Teile von Sätzen verknüpfen [Kürs93]. Man unterscheidet in anreihende Konjunktionen (*„und“*), disjunktive Konjunktionen (*„oder“*), temporale Konjunktionen (*„nachdem, solange, während“*) und durative Konjunktionen (*„solange“*). Weiterhin findet man Adverbien, wie Temporaladverbien (*„nie, währenddessen“*), Modaladverbien (*„anders, irgendwie, so“*) und Konditionaladverbien (*„dann“*).

Neben den aufgezeigten Schlüsselwörtern enthalten Anforderungen weitere konkrete, applikationsspezifische Formulierungen, die sich auf die Sensoren und Aktoren oder auf interne Zustände des Systems beziehen. Erst durch sie ergibt sich der eigentliche Inhalt einer Anforderung. Die Formulierung erfolgt durch Verben und Substantive, die den Satz vervollständigen. Die verwendeten Substantive werden nachfolgend als ***Nomen*** bezeichnet. Die verwendeten Verben repräsentieren die ***Werte***, die der hinter einem Nomen stehende Sensor oder Aktor annehmen kann. Dabei bezeichnen Vollverben eine Handlung, ein Geschehen oder einen Zustand [Flie86]. Hinzu kommen Hilfsverben, wie *„haben, sein, werden“*. Ausführliche Information zur Erstellung der Nomen und Werte können dem Abschnitt 4.4 entnommen werden.

Das Grundmuster einer Anforderung, die mit der Sicherheitsfachsprache formuliert wird, ist das Konditional, ein zusammengesetzter Satz. Dieser besteht aus zwei Tei-

len, die durch ein Komma voneinander getrennt sind. In einem Teilsatz wird eine Bedingung oder ein bestimmtes Ereignis formuliert wird (erkennbar am Schlüsselwort „*Wenn* ...“). Dieser Satzteil wird im Folgenden als *Bedingung B* bezeichnet. Im anderen Teilsatz wird eine zugehörige Reaktion formuliert (erkennbar am Schlüsselwort „*dann* ...“). Dieser Teilsatz wird als *Folgerung F* bezeichnet. Die Reihenfolge von Bedingung und Folgerung innerhalb des Satzes wird zunächst nicht festgelegt.

> Eine Anforderung A setzt sich aus einer *Bedingung B* und einer *Folgerung F* zusammen.
>
> Zwischen einer Bedingung und einer Folgerung besteht ein implikativer Zusammenhang.

Verbote, die lediglich einzelne Zustände des Systems beschreiben, die nie erreicht werden dürfen, können ebenfalls auf die gezeigte Satzstruktur überführt werden.

### *3.1.2 Definition des Gültigkeitsbereichs einer Anforderung*

Es wurde bereits festgestellt, dass es drei grundsätzliche Kategorien von Anforderungen an ein Steuerungsprogramm gibt:

a) ***Forderungen*** (demands - DE) beschreiben notwendige und erforderliche Reaktionen des Steuerungsprogramms auf festgelegte Ereignisse,

b) ***Verbote*** (prohibitions - PR) beschreiben untersagte Reaktionen des Steuerungsprogramms unter bestimmten Umständen,

c) ***Möglichkeiten*** (possibilities - PO) beschreiben erlaubte Reaktionen des Steuerungsprogramms unter bestimmten Umständen.

Eine Anforderung setzt sich aus einer Bedingung und einer Folgerung zusammen. Betrachtet man eine Bedingung und eine Folgerung unter Berücksichtigung des erzeugten System- bzw. Kontrollmodells, so lassen sich innerhalb des Zustandraumes des Systems einzelne oder mehrere Zustände identifizieren, bei denen die Werte der betrachteten Variablen mit den in den Bedingungen und Folgerungen definierten Variablenwerten übereinstimmen. Man spricht dann davon, dass diese Zustände die jeweilige Bedingung bzw. Folgerung erfüllen.

> Systemzustände, die eine Bedingung erfüllen, werden als *Startzustand S* bezeichnet.
>
> Systemzustände, die eine Folgerung erfüllen, werden als *Zielzustand Z* bezeichnet.
>
> Es wird ein *Beobachtungsintervall BI* definiert, das mit einem *Startzustand* beginnt und eine momentan nicht näher definierte Länge aufweist. Ein Beobachtungsintervall umfasst eine bestimmte Anzahl zeitlich aufeinander folgender Systemzustände.

Die Unterteilung der Anforderungen an ein Steuerungsprogramm erfolgt nachfolgend durch zwei Parameter:

1. der ***Modalparameter*** gibt die logische Bedeutung der Anforderung an, er stellt einen modalen Zusammenhang zwischen einem Zielzustand und einem Beobachtungsintervall her,
2. der ***Zeitparameter*** gibt die Länge eines Beobachtungsintervalls, d. h. die zeitliche Reichweite einer Anforderung an.

### *3.1.3 Ableitung des Modalparameters*

Zunächst wird eine Tabelle erstellt (Tabelle 3), in der die modalen Zusammenhänge zwischen dem Zielzustand Z (also der Erfüllung einer Folgerung) und dem Beobachtungsintervall BI dargestellt werden. Dabei erfolgt eine Betrachtung über die Existenz eines Zielzustandes sowohl innerhalb als auch außerhalb des Beobachtungsintervalls.

<table>
<tr><td colspan="2" rowspan="2"></td><th colspan="6">Innerhalb des Beobachtungsintervalls (BI)</th></tr>
<tr><th colspan="2">Forderung (muss)</th><th colspan="2">Möglichkeit (darf)</th><th colspan="2">Verbot (darf nicht)</th></tr>
<tr><th rowspan="6">Außerhalb des Beobachtungsintervalls</th><th rowspan="2">Forderung</th><td rowspan="2">I</td><td>innerhalb des BI: es muss ein Z geben</td><td rowspan="2">II</td><td>innerhalb des BI: es darf ein Z geben</td><td rowspan="2">III</td><td>innerhalb des BI: es darf kein Z geben</td></tr>
<tr><td>außerhalb des BI: es muss ein Z geben</td><td>außerhalb des BI: es muss ein Z geben</td><td>außerhalb des BI: es muss ein Z geben</td></tr>
<tr><th rowspan="2">Möglichkeit</th><td rowspan="2">IV</td><td>innerhalb des BI: es muss ein Z geben</td><td rowspan="2">V</td><td>innerhalb des BI: es darf ein Z geben</td><td rowspan="2">VI</td><td>innerhalb des BI: es darf kein Z geben</td></tr>
<tr><td>außerhalb des BI: es darf ein Z geben</td><td>außerhalb des BI: es darf ein Z geben</td><td>außerhalb des BI: es darf ein Z geben</td></tr>
<tr><th rowspan="2">Verbot</th><td rowspan="2">VII</td><td>innerhalb des BI: es muss ein Z geben</td><td rowspan="2">VIII</td><td>innerhalb des BI: es darf ein Z geben</td><td rowspan="2">IX</td><td>innerhalb des BI: es darf kein Z geben</td></tr>
<tr><td>außerhalb des BI: es darf kein Z geben</td><td>außerhalb des BI: es darf kein Z geben</td><td>außerhalb des BI: es darf kein Z geben</td></tr>
</table>

*Tabelle 3 - Matrix aus Modalkategorie und abstraktem Beobachtungsintervall*

Zu erkennen sind neun Kombinationstypen, von denen einige (grau unterlegt) jedoch nicht relevant bzw. untereinander redundant sind.

Bei den Kombinationen I, V und IX ist die Existenz des Zielzustandes in der gewählten Modalkategorie unabhängig vom Beobachtungsintervall. Der Zielzustand ist also unabhängig von einer aufgestellten Bedingung, d. h. die Folgerung würde im gesamten Zustandsraum des betrachteten Systems gültig sein. Die Kombinationen I, V und IX entfallen somit, weil sie nicht dem vorgegebenen Aufbau einer Anforderung (Bedingung und Folgerung) entsprechen. Andererseits können sie im Weiteren auch auf andere Kombinationstypen abgebildet werden (die Bedingung wäre dann generell TRUE).

Die Kombinationen II und IV, III und VII bzw. VI und VIII können durch Negation der Beobachtungsintervalle auf die jeweils andere Kombination abgebildet werden, sie sind also redundant. Aus diesem Grund entfallen die Kombinationen II und III. Die Kombination VI wird jedoch trotzdem beibehalten, da sich hier später auf sprachlicher Ebene unterschiedliche Formulierungen ergeben werden.

Nach Abzug der ausgeschlossenen Kombinationen bleiben lediglich vier Varianten übrig, diese stellen die „automatisierungstechnisch stark relevanten“ Anforderungen an Steuerungsprogramme dar.

IV - Einfache Forderung – DEs (simple demand)

VI - Einfaches Verbot – PRs (simple prohibition)

VII - Erweiterte Forderung – DEe (extended demand)

VIII - Erweiterte Möglichkeit – POe (extended possibility)

### *3.1.4 Ableitung des Zeitparameters*

Neben der Kategorisierung der SPS-Anforderungen hinsichtlich ihrer Bedeutung besteht ein weiteres Unterscheidungsmerkmal innerhalb dieser Kategorien in der *Reichweite* der jeweiligen Aussage. Mit ihr wird spezifiziert, über welchen Zeitraum die entsprechende Anforderung gültig sein soll.

Es wurde bereits ein Beobachtungsintervall definiert, in dem je nach Modalparameter ein festgelegter Zielzustand entweder existieren muss, existieren darf oder nicht existieren darf. Das Beobachtungsintervall beginnt immer mit einem Startzustand und besteht aus einer Menge aufeinander folgender Zustände. Die Länge eines Beobachtungsintervalls, d. h. die Anzahl der aufeinander folgenden relevanten Zustände, wird durch den Zeitparameter festgelegt.

| |
|---|
| Ein *Beobachtungsintervall BI* beginnt mit einem *Startzustand S* und endet mit einem *Endzustand E*. |

Typische automatisierungstechnische Anforderungen an ein Steuerungsprogramm beziehen sich auf zwei Zeitebenen: tritt auf eine Bedingung die zugehörige Reaktion innerhalb einer vorgegebenen (möglichst kurzen) Zeitspanne ein (kurzfristige, operative Anforderungen) bzw. tritt auf eine Bedingung die zugehörige Reaktion überhaupt ein (langfristige, strategische Anforderungen)?

Ausgehend von dieser Tatsache lassen sich zwei Hauptgruppen mit insgesamt fünf Varianten von Beobachtungsintervallen differenzieren. Die beiden ersten Varianten

beziehen sich auf die typische Arbeitsweise einer speicherprogrammierbaren Steuerung. Der Zeitpunkt des Endzustandes und somit auch die Länge des Beobachtungsintervalls ist an den SPS-Zyklus gekoppelt.

- ***Zustand***: das Beobachtungsintervall ist auf alle Zustände begrenzt, die im selben SPS-Zyklus liegen,
- ***Direkt***: das Beobachtungsintervall ist auf alle Zustände beschränkt, die im selben und im direkt nachfolgenden SPS-Zyklus liegen,

Die Länge der folgenden drei Varianten der Beobachtungsintervalle ist im Gegensatz zu den beiden vorherigen nicht starr festgelegt. Der Zeitpunkt des Endzustandes wird durch den Bedingungsteil der Anforderung gegeben.

- ***Selbstbegrenzt***: Der Startzustand ist hierbei wie üblich mit der Erfüllung der Bedingung gegeben (*S* entspricht dem Zustand, in dem *B* erfüllt wird). Der Endzustand ist gegeben, sobald die definierte Bedingung erstmals nicht mehr erfüllt ist (*E* entspricht dem Zustand, in dem *B* nicht mehr erfüllt wird).
- ***Fremdbegrenzt***: Das Beobachtungsintervall wird vom Startzustand eingeleitet (*S* entspricht dem Zustand, in dem $B_1$ erfüllt wird), der Endzustand ist durch die Erfüllung einer zusätzlichen Bedingung gegeben (*E* entspricht dem Zustand, in dem $B_2$ erfüllt wird). Die Bedingung $B_1$ muss dabei innerhalb des Beobachtungsintervalls nicht unbedingt wahr bleiben.
- ***Unbegrenzt***: Das Beobachtungsintervall wird ebenfalls vom Startzustand eingeleitet, hat jedoch kein Ende. Es handelt sich hierbei um ein offenes Intervall.

Die Abbildung 10 soll diese fünf Varianten des Beobachtungsintervalls noch einmal verdeutlichen.

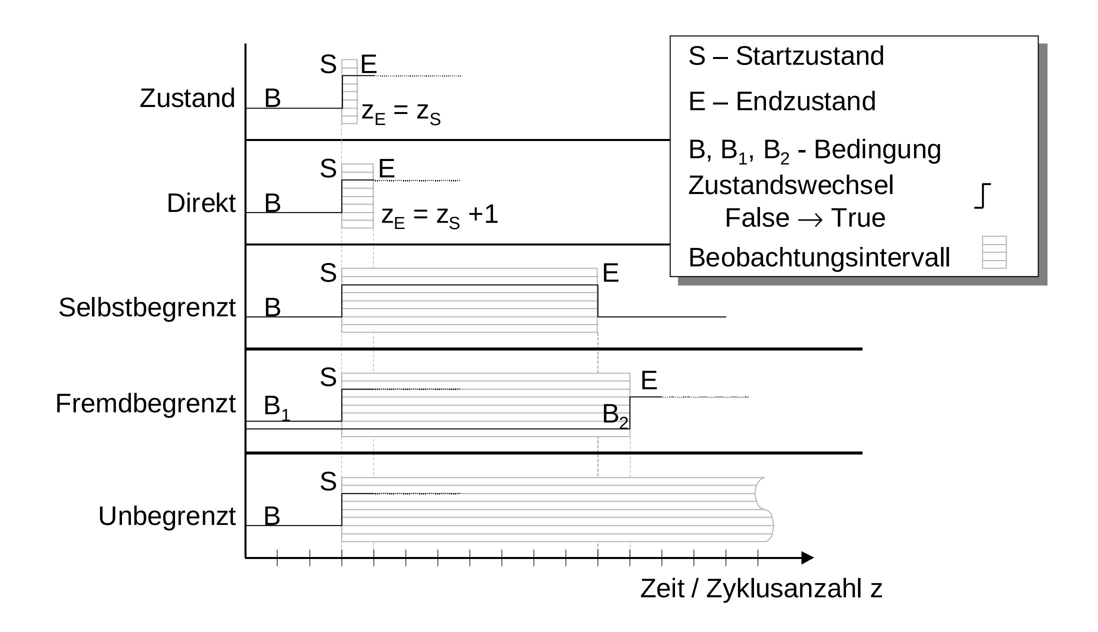



*Abbildung 10 - Definition der Beobachtungsintervalle*

### *3.1.5 Anforderungsmatrix*

Für die Zusammenstellung der verschiedenen Arten von Anforderungen an ein Steuerungsprogramm können nun der Modalparameter und der Zeitparameter in einer Matrix vereinigt werden.

| **Einfache Forderung** | | **Einfaches Verbot** |
|---|---|---|
| DEs1 - Zustand | Nicht definiert | PRs1 - Zustand |
| DEs2 - Direkt | | Nicht definiert |
| DEs3 - Selbstbegrenzt | | PRs3 - Selbstbegrenzt |
| DEs4 - Fremdbegrenzt | | PRs4 - Fremdbegrenzt |
| DEs5 - Unbegrenzt | | PRs5 - Unbegrenzt |
| **Erweiterte Forderung** | **Erweiterte Möglichkeit** | |
| DEe1 - Zustand | POe1 - Zustand | Nicht definiert |
| DEe2 - Direkt | Nicht definiert | |
| DEe3 - Selbstbegrenzt | POe3 - Selbstbegrenzt | |
| DEe4 - Fremdbegrenzt | POe4 - Fremdbegrenzt | |
| DEe5 - Unbegrenzt | POe5 - Unbegrenzt | |

*Tabelle 4 - Matrix aus automatisierungstechnisch relevanten Kategorien und Beobachtungsintervallen*

Die nichtrelevanten Kategorien wurden in der Tabelle 4 wiederum grau hinterlegt. Da die Formulierung von direkten Beobachtungsintervallen nur für Forderungen sinnvoll ist, werden letztendlich durch den Modal- und den Zeitparameter insgesamt 18 Anforderungskategorien definiert.

Diese 18 Anforderungskategorien unterscheiden sich jedoch auch stark hinsichtlich ihrer Relevanz bzw. ihren Einsatzmöglichkeiten bei der Spezifikation eines Steuerungsprogramms. Sie umfassen neben den bereits dargestellten Anforderungen aus der Automatisierungstechnik auch Eigenschaftsklassen, wie sie in der Informatik bekannt und üblich sind. Zu diesen gehören z. B. Fortschritts-, Lebendigkeits- und Sicherheitseigenschaften, die sich in der Matrix nach Tabelle 5 wiederfinden lassen.

| **Thematik** | **Darstellung mit SFS** | **Bezeichnung in der Informatik** |
|---|---|---|
| Sollverhalten der Steuerung - kurzfristig | DEs1, DEs2, DEe1, DEe2 | Fortschrittseigenschaften |
| Sollverhalten der Steuerung - langfristig | DEs3 ... DEs5, DEe3 ... DEe5 | Lebendigkeitseigenschaften |
| auszuschließende Reaktionen | PRs1 ... PRs5, eingeschränkt auch POe1 ... POe5 | Sicherheitseigenschaften |

*Tabelle 5 - Verwendung der Kategorien der Sicherheitsfachsprache*

### *3.1.6 Grafische Repräsentation der SFS-Kategorien*

Der nun folgende Abschnitt fasst die zuvor zusammengestellten Kategorien der Sicherheitsfachsprache nochmals zusammen und stellt die Zusammenhänge zwischen Bedingung und Folgerung in diesen Kategorien grafisch in ihren zeitlichen Abläufen dar. Dieser zeitliche Zusammenhang ist anhand der ablaufenden SPS-Zyklen nachzuvollziehen.

Die Abbildung 11 zeigt, wie die nachfolgenden Diagramme zu interpretieren sind.




*Abbildung 11 - Legende für die folgenden Timingdiagramme*

Mit B, B1 und B2 sind einzelne Bedingungen oder auch mehrere konjunktiv verknüpfte Bedingungen gemeint, F bezeichnet eine einzelne Folgerung oder mehrere konjunktiv verknüpfte Folgerungen. Diese Bedingungen bzw. Folgerungen können zum aktuellen Zeitpunkt (identifizierbar durch den SPS-Zyklus) entweder erfüllt („True“) oder nicht erfüllt („False“) sein. Darüber hinaus besteht jedoch auch die Möglichkeit, dass der beschriebene Zusammenhang nicht mit einem eindeutigen Zustand der Bedingung oder Folgerung verknüpft ist.

Die Abbildung 12 zeigt zunächst die einzelnen Diagramme für die vier Anforderungskategorien mit dem Zeitparameter „Zustand“.

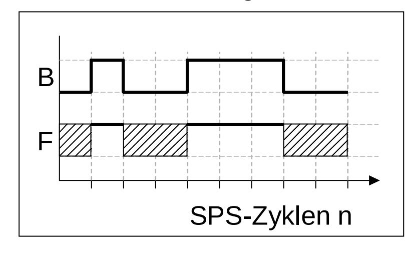
So wird für die einfache Forderung mit diesem Zeitparameter (DEs1) verlangt, dass in jedem SPS-Zyklus, in dem die Bedingung B wahr ist, auch die Folgerung F erfüllt sein muss. Ist die Bedingung nicht erfüllt, so kann der Zustand der Folgerung beliebig sein. Erst bei der erweiterten Forderung (DEe1) wird zusätzlich verlangt, dass bei nichterfüllter Bedingung B auch die Folgerung nicht erfüllt sein darf.

Die Interpretation der weiteren Diagramme (Abbildung 13, Abbildung 14, Abbildung 15, Abbildung 16) ergibt sich auf analoge Weise. Hier sind besonders deutlich die Zusammenhänge der einzelnen Anforderungskategorien zu erkennen:

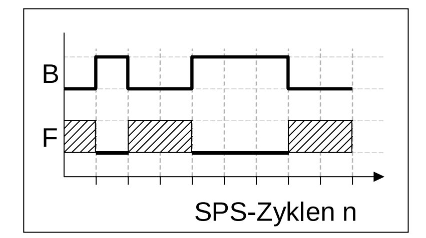
- das einfache Verbot stellt eine Umkehrung der einfachen Forderung dar,
- die erweiterte Möglichkeit ergibt sich durch Invertierung des einfachen Verbotes,
- die erweiterte Forderung ergibt sich durch Verschmelzung der einfachen Forderung mit der erweiterten Möglichkeit.

Diese Zusammenhänge werden nachfolgend auch bei der Erstellung der temporallogischen Formeln wieder aufgegriffen (siehe Abschnitt 3.2.4).

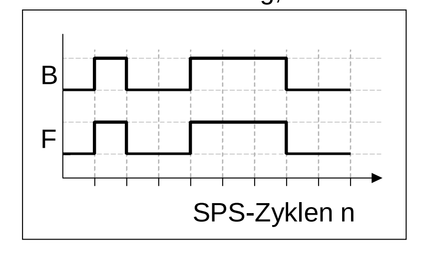
einfache Forderung, Zustand




einfaches Verbot, Zustand




erweiterte Forderung, Zustand




erweiterte Möglichkeit, Zustand

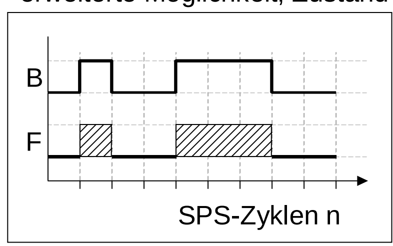
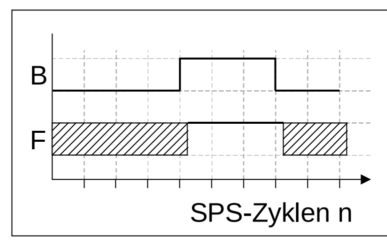
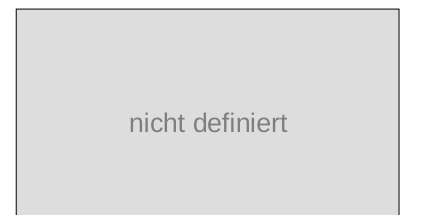



*Abbildung 12 - Timingdiagramme „Zustand“*

einfache Forderung, direkt




erweiterte Forderung, direkt

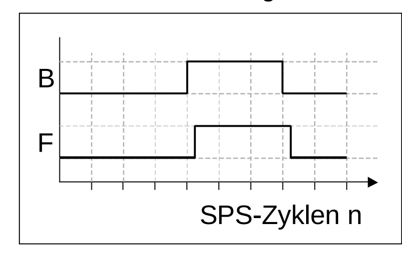
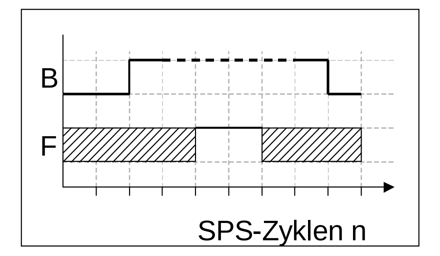




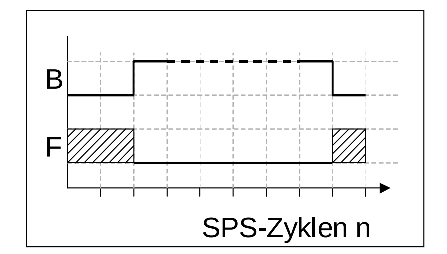


*Abbildung 13 - Timingdiagramme „Direkt“ (nur Forderungen sind definiert)*

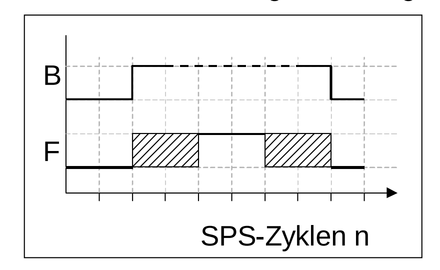
einfache Forderung, selbstbegrenzt




einfaches Verbot, selbstbegrenzt




erweiterte Forderung, selbstbegrenzt




erweiterte Möglichkeit, selbstbegrenzt

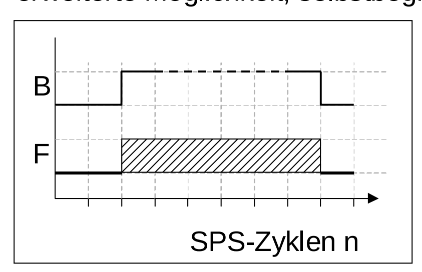
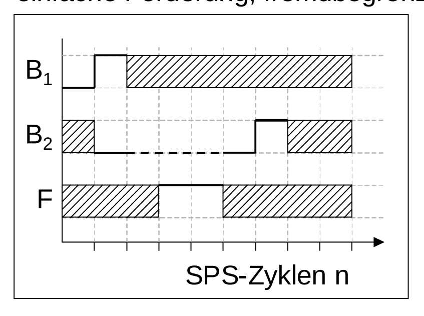
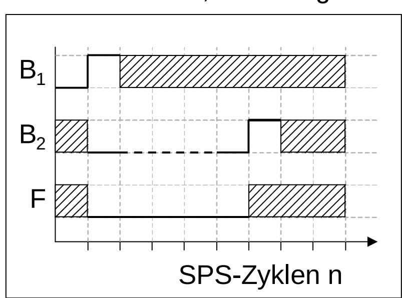



*Abbildung 14 - Timingdiagramme „Selbstbegrenzt“*

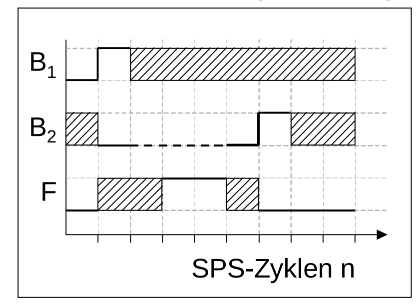
einfache Forderung, fremdbegrenzt




einfaches Verbot, fremdbegrenzt




erweiterte Forderung, fremdbegrenzt




erweiterte Möglichkeit, fremdbegrenzt

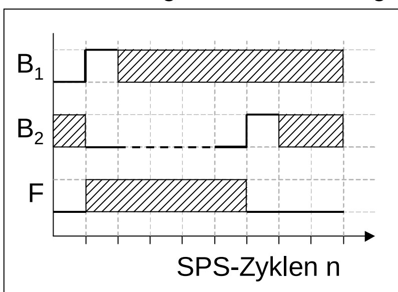
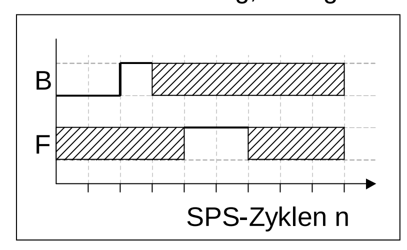
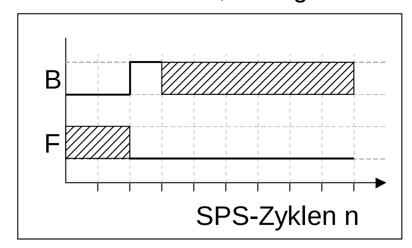



*Abbildung 15 - Timingdiagramme „Fremdbegrenzt“*

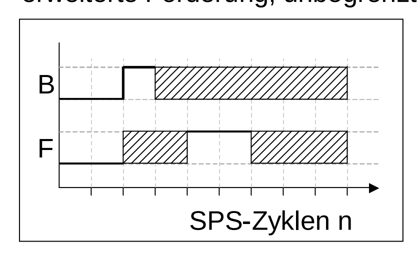
einfache Forderung, unbegrenzt




einfaches Verbot, unbegrenzt




erweiterte Forderung, unbegrenzt




erweiterte Möglichkeit, unbegrenzt

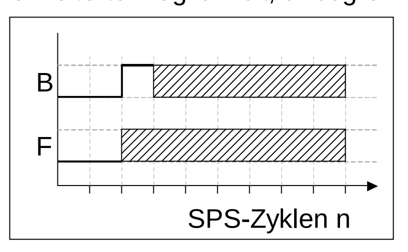



*Abbildung 16 - Timingdiagramme „Unbegrenzt“*

## 3.2 Temporale Logik als formale Basis der SFS

Die zuvor beschriebenen Anforderungskategorien müssen formal dargestellt werden. Für die zuvor beschriebene Einbindung der Sicherheitsfachsprache in eine Verifikationsumgebung wird eine Darstellung der Programmanforderungen in Form von Formeln der Temporalen Logik benötigt. Die folgenden Abschnitte geben einen Überblick über die logischen Grundlagen und leiten die notwendigen Darstellungsformen ab.

### *3.2.1 Aussagenlogik*

Das Alphabet der Aussagenlogik besteht aus atomaren Formeln und Symbolen [Rupp96]. Zu den Symbolen gehören:

$\neg$ ≡ Negation – *„nicht“*

$\rightarrow$ ≡ Konditional – *„wenn ... dann“* ($\neg p \vee q$)

( ≡ öffnende Klammer

) ≡ schließende Klammer

Formeln werden durch folgende Regeln gebildet:

Jede atomare Formel ist eine Formel.

Wenn *p* eine Formel ist, dann ist $\neg p$ auch eine Formel.

Wenn *p* und *q* Formeln sind, dann ist $p \rightarrow q$ auch eine Formel.

Mit diesen Grundlagen lassen sich andere Operatoren und Symbole ableiten:

$p \wedge q \equiv \neg (p \rightarrow \neg q)$

$p \vee q \equiv \neg p \rightarrow q$

$p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$

$TRUE \equiv p \vee \neg p$

$FALSE \equiv \neg\ TRUE$

### *3.2.2 Temporale Logik*

Die Temporale Logik (TL) ist eine um zeitliche Operatoren erweiterte Aussagenlogik. Sie dient als Werkzeug, um zeitliche Besonderheiten in der logischen Struktur eines Programms beschreiben zu können [Fish93], [Krög87], [Mann92], [Mann95], [Emer90], [Clar86], [Mord83]. Die Temporale Logik kann zur Programmverifikation verwendet werden. Man formalisiert hierbei den Begriff des Zeitpunktes und betrachtet nicht mehr feste Interpretationen, sondern vielmehr Interpretationen, die zu verschiedenen Zeitpunkten einen unterschiedlichen Wahrheitsgehalt haben können. So können bestimmte Aussagen in Abhängigkeit vom aktuellen Zeitpunkt wahr oder unwahr sein.

| Zeitpunkt | Aktion | a | b | c | $A = [(a + b = c \wedge a > 0) \rightarrow b > 0]$ |
|---|---|---|---|---|---|
| 1 | ... | 3 | -3 | 0 | FALSE |
| 2 | c := b | 3 | -3 | -3 | TRUE |
| 3 | b := b - a | 3 | -6 | -3 | FALSE |

*Tabelle 6 - Abhängigkeit des Wahrheitsgehaltes einer Aussage vom Zeitpunkt der Betrachtung*

In der oben stehenden Tabelle 6 [Krög87] ist die Aussage A „Wenn die Summe aus a und b gleich c ist und a ist größer als Null, dann ist b größer als Null“ für die Zeitpunkte 1 und 3 falsch, während sie im Zeitpunkt 2 richtig war.

Für die Darstellung solcher Besonderheiten existieren in der Temporalen Logik spezielle Operatoren (Tabelle 7), die sich auf zeitliche Abhängigkeiten beim Ablauf eines Programms beziehen.

| **Operator/ Symbol** | **Bezeichnung** | **Bedeutung (Bezug zum Referenzzeitpunkt)** |
|---|---|---|
| X / ❍ | „ne**x**ttime”-Operator | „im nächsten Zeitpunkt gilt” |
| F / ◊ | „eventually / sometime / **f**inally”-Operator | „irgendwann gilt schließlich” |
| G / ☐ | „henceforth / always / globally / **g**enerally”-Operator | „von nun an gilt immer” |
| U / ∪ | „**u**ntil”-Operator | „x gilt solange, bis y gilt” |

*Tabelle 7 - Beispiele für Temporaloperatoren*

### *3.2.3 Computation Tree Logic*

Innerhalb der Temporalen Logik sind zwei Sichtweisen der Zeit möglich:

1) Lineare Sichtweise: Hierbei gibt es für jeden Zeitpunkt nur eine mögliche Zukunft, Verzweigungen zu alternativen Varianten sind nicht möglich (linear TL).
2) Verzweigte Sichtweise: Hierbei gibt es an bestimmten Zeitpunkten die Möglichkeit, dass sich die Zeit durch Nichtdeterminismus verzweigt. Es gibt mehrere Alternativen der Zukunft (branching-time TL).

Um Zustandssequenzen eines Systems beschreiben zu können, benötigt man eine Logik, die es gestattet, die relativen Zeitbegriffe der Sicherheitsfachsprache (*„sofort“, „nie“, „solange“*) zu verwenden. Die Wahl fiel auf CTL (Computation Tree Logic), eine Temporale Logik, die sich an der zweiten Sichtweise orientiert.

Neben den bereits erwähnten TL-Operatoren gibt es für branching-time-Logiken die zusätzlichen Pfad- bzw. Wegquantoren A (on all paths) bzw. E (on some paths).

Damit ergeben sich die folgenden Grundoperationen

$EX\ \varphi$ - in mindestens einem Pfad, der dem aktuellen Zustand folgt, wird $\varphi$ im unmittelbar nachfolgenden Zustand gültig werden

$AX\ \varphi$ - $\varphi$ ist in allen unmittelbar nachfolgenden Zuständen gültig

$EF\ \varphi$ - in mindestens einem Pfad, der dem aktuellen Zustand folgt, wird $\varphi$ irgendwann gültig werden

$AF\ \varphi$ - in allen Pfaden, die dem aktuellen Zustand folgen, wird $\varphi$ irgendwann gültig werden

$EG\ \varphi$ - in mindestens einem Pfad, der dem aktuellen Zustand folgt, wird $\varphi$ immer gültig sein

$AG\ \varphi$ - $\varphi$ ist in allen nachfolgenden Zuständen gültig

$E\ [\varphi_1\ U\ \varphi_2]$ - in mindestens einem Pfad, der dem aktuellen Zustand folgt, gilt $\varphi_1$ solange, bis $\varphi_2$ gilt

$A\ [\varphi_1\ U\ \varphi_2]$ - in allen Pfaden, die dem aktuellen Zustand folgen, gilt $\varphi_1$ solange, bis $\varphi_2$ gilt,

Für CTL-Formeln gelten alle Äquivalenzen der Aussagenlogik ([Mann95], [Dwye98]).

$EX\varphi \equiv \neg AX\neg\varphi$, $AX\varphi \equiv \neg EX\neg\varphi$,

$EF\varphi \equiv \neg AG\neg\varphi$, $AF\varphi \equiv \neg EG\neg\varphi$, $EG\varphi \equiv \neg AF\neg\varphi$, $AG\varphi \equiv \neg EF\neg\varphi$,

$EF\varphi \equiv E[TRUE\ U\ \varphi]$, $AF\varphi \equiv A[TRUE\ U\ \varphi]$,

$E[\varphi_1\ U\ \varphi_2] \equiv \varphi_2 \vee (\varphi_1 \wedge EX\ E[\varphi_1\ U\ \varphi_2])$, $A[\varphi_1\ U\ \varphi_2] \equiv \varphi_2 \vee (\varphi_1 \wedge AX\ A[\varphi_1\ U\ \varphi_2])$

Die oben dargestellten Umwandlungsoperationen erlangen im Zusammenhang mit einer natürlichsprachlichen Übersetzung dieser Gleichungen eine besondere Bedeutung. So kann die Gleichung $EF\varphi = \neg AG \neg\varphi$ auch folgendermaßen interpretiert werden:

> „Wenn in mindestens einem Pfad, der dem aktuellen Zustand folgt, φ irgendwann gültig wird, dann ist es nicht wahr, dass φ in allen nachfolgenden Zuständen ungültig ist."

Die inhaltlichen Äquivalenzen, die bei komplexen sprachlichen Formulierungen nicht immer sofort zu erkennen sind, lassen sich also auf temporallogischer Ebene eindeutig nachweisen.

### *3.2.4 Darstellung der Anforderungskategorien in CTL-Formeln*

Es folgt nun die ausführliche Darstellung der Umsetzung der einzelnen Anforderungskategorien der Sicherheitsfachsprache, die in Abschnitt 3.1.5 abgeleitet wurden, in CTL-Formeln. Bevor dies geschehen kann, müssen jedoch noch innerhalb des Systemmodells Beobachtungsvariablen definiert werden, um die zyklische Ausführung der einzelnen Abschnitte des SPS-Zyklus (Eingänge lesen, Programm bearbeiten, Ausgänge schreiben) beobachten zu können. (vgl. Abbildung 17 und Abschnitt 2.3.2).

| **Beobachtungsvariable** | **Bedeutung** |
|---|---|
| rdy_in | Einlesen der Eingangsvariablen ist abgeschlossen |
| rdy_plc | Bearbeitung des SPS-Programms ist abgeschlossen |
| rdy_out | Ausgeben der Ausgangsvariablen ist abgeschlossen |
| rdy_env | Berechnung des Umgebungsmodells ist abgeschlossen |

*Tabelle 8 - Definition von Beobachtungsvariablen*

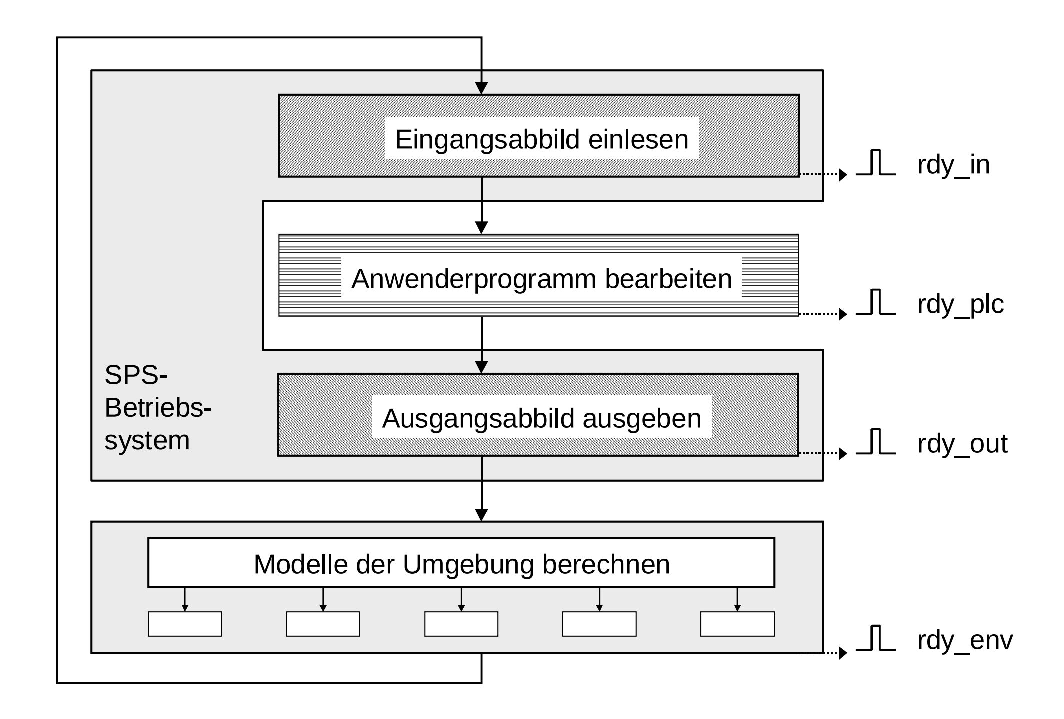



*Abbildung 17 - Beobachtungsvariablen zur Identifikation des Systemzustandes*

Der Abschluss eines einzelnen Abschnittes ist am kurzzeitigen TRUE-Zustand der entsprechenden Beobachtungsvariablen zu erkennen, für die restliche Zeit stehen diese Variablen immer auf FALSE.

Für die Umsetzung der Anforderungen in die CTL-Formeln gelten folgende generelle Bedingungen:

- Die aufgestellten Anforderungen gelten immer für den gesamten Erreichbarkeitsraum des Systems. Diesem Umstand muss durch die Benutzung der CTL-Operatoren AG (‚generell gilt in allen nachfolgenden Zuständen') Rechnung getragen werden. Wenn der Gültigkeitsbereich einer Anforderung nur für bestimmte Abschnitte (Sequenzen) gelten soll, so ist dieser Bereich durch die aufgestellten Bedingungen bzw. das Beobachtungsintervall zu beschränken.

- Alle Anforderungen, die sich auf die korrekte Ausführung des SPS-Programms beziehen, müssen mit den eingeführten Beobachtungsvariablen verknüpft werden. Viele Analysen sind erst dann sinnvoll durchführbar, wenn die Bearbeitung des SPS-Zyklus abgeschlossen ist (erkennbar durch ‚rdy_plc = TRUE').

- Die kürzeste Reaktionszeit einer realen SPS ist von ihrer Zykluszeit und der zeitlichen Abfolge ihrer Betriebssystemroutinen abhängig (vergleiche Abbildung 8 und Abbildung 10). Die schnellste Reaktion einer SPS ist somit dann gegeben, wenn ein relevantes Ereignis im Moment des Einlesens der Sensorsignale eintritt (rdy_in) und dieses Ereignis unmittelbar anschließend im Steuerungsprogramm verarbeitet wird (rdy_plc wird erstmalig gesetzt). Durch diese beiden Beobachtungsvariablen kann diese Tatsache überwacht werden.

Im Abschnitt 3.1.5 wurden die folgenden Kategorien des Modalparameters abgeleitet:

| **Einfache Forderung** | **Einfaches Verbot** |
|---|---|
| innerhalb des *BI* **muss** es ein *F* geben<br>außerhalb des *BI* darf es ein *F* geben | innerhalb des *BI* **darf** es **kein** *F* geben<br>außerhalb des *BI* darf es ein *F* geben |
| **Erweiterte Forderung** | **Erweiterte Möglichkeit** |
| innerhalb des *BI* **muss** es ein *F* geben<br>außerhalb des *BI* **darf** es **kein** *F* geben | innerhalb des *BI* darf es ein *F* geben<br>außerhalb des *BI* **darf** es **kein** *F* geben |

An dieser Stelle lassen sich auch wieder Verbindungen zwischen den Kategorien des Modalparameters ableiten:

- Bei der einfachen Forderung muss es innerhalb des Beobachtungsintervalls ein *F* geben.
- Das einfache Verbot ist eine Umkehrung der einfachen Forderung (innerhalb des Beobachtungsintervalls darf es kein *F* geben).
- Bei der erweiterten Möglichkeit wird das Beobachtungsintervall im Vergleich zum einfachen Verbot invertiert (es darf außerhalb des Beobachtungsintervalls kein *F* geben).
- Die erweiterte Forderung kann man als Verknüpfung zwischen der einfachen Forderung (innerhalb des *BI* muss es ein *F* geben) und der erweiterten Möglichkeit (außerhalb des *BI* darf es kein *F* geben) betrachten.

Diese Zusammenhänge spiegeln sich auch nachfolgend in den CTL-Formeln wieder. Die folgende Auflistung der Anforderungskategorien mit ihrer CTL-Darstellung ist durch den Zeitparameter gegliedert:

- Zuerst werden die Kategorien mit dem Zeitparameter „Zustand“ dargestellt,
- danach wird die Umsetzung der beiden Kategorien mit dem Zeitparameter „Direkt“ gezeigt,
- im Gegensatz zum Zeitparameter „Direkt“ wird die Erfüllung der Folgerung bei Kategorien mit dem Zeitparameter „Unbegrenzt“ nicht im nächsten SPS-Zyklus, sondern lediglich irgendwann danach gefordert,
- bei Kategorien mit dem Zeitparameter „Fremdbegrenzt“ wird der Zeitraum für die Erfüllung der Folgerung jedoch wieder durch eine zweite, unabhängige Bedingungen eingegrenzt,
- bei Kategorien mit dem Zeitparameter „Selbstbegrenzt“ ist das Ende des Beobachtungsintervalls nicht durch eine zweite Bedingung, sondern durch die Nichterfüllung der ursprünglichen Bedingung gegeben.

Es folgt nun die formale Beschreibung der Menge der syntaktisch korrekten CTL-Formeln und deren Bedeutung. Die Darstellung orientiert sich an der eingeführten Terminologie für Registernetze [Hein01], die als Basis des Systemmodells dienen.

**Zeitparameter „Zustand“**

### Kategorie DEs1 - einfache Forderung - Zustand

Analyseinhalt: In jedem Zustand, in dem rdy_plc und eine Bedingung *B* erfüllt ist, muss gleichzeitig auch eine Folgerung *F* erfüllt sein.

$$AG\ (\ (\text{rdy_plc} \wedge B) \rightarrow F\ )$$

### Kategorie PRs1 - einfaches Verbot - Zustand

Analyseinhalt: In jedem Zustand, in dem rdy_plc und eine Bedingung *B* erfüllt ist, darf eine Folgerung *F* nicht erfüllt sein.

$$AG\ (\ (\text{rdy_plc} \wedge B) \rightarrow \neg F\ )$$

Diese Formel lässt sich umformen, sie erhält dadurch das typische Aussehen einer Sicherheitseigenschaft.

$$AG\ \neg(\ \text{rdy_plc} \wedge B \wedge F\ )$$

### Kategorie POe1 - erweiterte Möglichkeit - Zustand

Analyseinhalt: Hier wird verlangt, dass eine bestimmte Folgerung nur dann eintreten darf, wenn gleichzeitig eine bestimmte Bedingung erfüllt wird. In der Umkehrung bedeutet dies, dass in jedem Zustand, in dem zwar rdy_plc aber eine Bedingung *B* nicht erfüllt ist, die Folgerung *F* nicht erfüllt sein darf.

$$AG\ (\ (\text{rdy_plc} \wedge F) \rightarrow B\ )\ \text{bzw.}\ AG\ (\ (\text{rdy_plc} \wedge \neg B) \rightarrow \neg F\ )$$

Nach Umformung erhält man:

$$AG\ \neg(\ \text{rdy_plc} \wedge \neg B \wedge F\ )$$

### Kategorie DEe1 - erweiterte Forderung - Zustand

Analyseinhalt: In jedem Zustand, in dem rdy_plc und eine Bedingung *B* erfüllt ist, muss auch eine Folgerung *F* erfüllt sein. Zusätzlich darf in jedem Zustand, in dem zwar rdy_plc aber die Bedingung *B* nicht erfüllt ist, die Folgerung *F* nicht erfüllt sein.

$$AG\ (\ (\ (\text{rdy_plc} \wedge B) \rightarrow F\ ) \wedge (\ (\text{rdy_plc} \wedge \neg B) \rightarrow \neg F\ )\ )$$

Die gezeigte Formel lässt sich umformen und vereinfachen.

$$AG\ (\ (rdy_plc \wedge B) \leftrightarrow (rdy_plc \wedge F\ )\ )$$

$$AG\ (\ rdy_plc \rightarrow (B \leftrightarrow F\ )\ )$$

### Zeitparameter „Direkt“

#### Kategorie DEs2 - einfache Forderung - Direkt

Analyseinhalt: Nach jedem Zustand, in dem rdy_in und eine Bedingung *B* erfüllt ist, muss im nächstmöglichen Zustand, in dem rdy_plc erfüllt ist, auch eine Folgerung *F* erfüllt sein. Es besteht die Forderung, dass das Steuerungsprogramm auf ein beim Einlesen der Sensorwerte erkanntes Ereignis unmittelbar reagiert. Diese Forderung ist erfüllt, wenn die Reaktion nach der unmittelbar folgenden Abarbeitung des SPS-Programms eingetreten ist, d. h. sobald die Beobachtungsvariable rdy_plc erstmals von FALSE auf TRUE gewechselt hat.

$$AG\ (\ (rdy_in \wedge B) \rightarrow A[\ \neg rdy_plc\ U\ (rdy_plc \wedge F)\ ]\ )$$

Die CTL-Formel enthält den Until-Operator U, durch die Verbindung mit ¬rdy_plc und rdy_plc wird die steigende Signalflanke der Beobachtungsvariablen erkannt. Die Formel kann folgendermaßen interpretiert werden: Wenn nach dem Einlesen der Sensorwerte eine Bedingung *B* erfüllt ist, muss die darauf folgende Phase der Bearbeitung des SPS-Programm immer die zugehörige Reaktion *F* liefern.

#### Kategorie DEe2 - erweiterte Forderung - Direkt

Analyseinhalt: Nach jedem Zustand, in dem rdy_in und eine Bedingung *B* erfüllt ist, muss im nächsten Zustand, in dem rdy_plc erfüllt ist, auch eine Folgerung *F* erfüllt sein. In Erweiterung zur Kategorie DEs2 gilt innerhalb dieser Anforderungskategorie, dass unmittelbar nach denjenigen Zuständen, in denen zwar rdy_in, jedoch nicht die bezeichnete Bedingung *B* erfüllt ist, die genannte Folgerung auch nicht erfüllt sein darf.

$$\begin{aligned} AG\ (\ (\ (rdy_in \wedge B) &\rightarrow A[\ \neg rdy_plc\ U\ (rdy_plc \wedge F)\ ]\ ) \\ \wedge\ (\ (rdy_in \wedge \neg B) &\rightarrow A[\ \neg rdy_plc\ U\ (rdy_plc \wedge \neg F)\ ]\ )\ ) \end{aligned}$$

### Zeitparameter „Unbegrenzt“

#### Kategorie DEs5 - einfache Forderung - Unbegrenzt

Analyseinhalt: Nach jedem Zustand, in dem rdy_in und eine Bedingung *B* erfüllt ist, muss es irgendwann einen Zustand geben, in dem rdy_plc und *F* erfüllt sind. Es besteht die Forderung, dass das Steuerungsprogramm auf ein beim Einlesen der Sensorwerte erkanntes Ereignis immer irgendwann reagiert. Die auslösende Bedingung muss dann jedoch nicht mehr erfüllt sein. Das Beobachtungsintervall beginnt mit dem

Zustand, in dem beim Einlesen der Sensorwerte erstmalig die Bedingung erfüllt ist.

$$AG\ (\ (rdy_in \wedge B) \rightarrow AF\ (rdy_plc \wedge F\ )\ )$$

**Kategorie PRs5 - einfaches Verbot - Unbegrenzt**

Analyseinhalt: Sobald es einen Zustand gibt, in dem rdy_in und eine Bedingung *B* erfüllt werden, darf es danach keinen Zustand mehr geben, in dem rdy_plc und eine Folgerung *F* erfüllt werden. Diese Kategorie ist wiederum vergleichbar mit der Kategorie DEs5 - einfache Forderung - unbegrenzt, nur dass hierbei nach dem Erkennen des Ereignisses die beschriebene Reaktion niemals wieder eintreten darf.

$$AG\ (\ (rdy_in \wedge B) \rightarrow \neg EF\ (rdy_plc \wedge F\ )\ )$$

bzw. $$AG\ (\ (rdy_in \wedge B) \rightarrow AG\ \neg(rdy_plc \wedge F\ )\ )$$

**Kategorie POe5 - erweiterte Möglichkeit - Unbegrenzt**

Analyseinhalt: Bei dieser Kategorie wird gefordert, dass eine bestimmte Folgerung erst dann ausgeführt werden darf, wenn vorher eine bestimmte Bedingung erfüllt wurde. Das Beobachtungsintervall beginnt also mit der erstmaligen Erfüllung der Bedingung. Kehrt man diese Anforderung wieder um, so bedeutet dies, dass die Folgerung solange nicht ausgeführt werden darf, bis die Bedingung nicht mindestens einmal erfüllt wurde. D. h. außerhalb des Beobachtungsintervalls darf es keinen Zustand geben, in dem (rdy_plc $\wedge$ *F*) wahr ist.

$$A[\ \neg(rdy_plc \wedge F)\ U\ (rdy_in \wedge B)\ ]$$

**Kategorie DEe5 - erweiterte Forderung - Unbegrenzt**

Analyseinhalt: Nach jedem Zustand, in dem rdy_in und eine Bedingung *B* erfüllt ist, muss es irgendwann einen Zustand geben, in dem rdy_plc und F erfüllt sind. Zusätzlich darf es keinen Zustand mit rdy_plc und *F* geben, bevor nicht rdy_in und eine Bedingung *B* erfüllt worden sind. Dieses Verhalten lässt sich aus den beiden Kategorien DEs5 und POe5 zusammensetzen. Einerseits besteht die Forderung, dass das Steuerungsprogramm auf ein beim Einlesen der Sensorwerte erkanntes Ereignis immer irgendwann reagiert, andererseits darf die genannte Reaktion auch nicht vorher eintreten.

$$AG\ (\ (\ (rdy_in \wedge B) \rightarrow AF\ (rdy_plc \wedge F\ )\ ) \\ \wedge A[\ \neg(rdy_plc \wedge F)\ U\ (rdy_in \wedge B)\ ]\ )$$

**Zeitparameter „Fremdbegrenzt"**

### Kategorie DEs4 - einfache Forderung - Fremdbegrenzt

Analyseinhalt: Sobald es einen Zustand gibt, in dem rdy_in und eine Startbedingung $B_1$ erfüllt werden, muss es danach immer einen Zustand geben, in dem rdy_plc und eine Folgerung F erfüllt werden, bevor es einen Zustand gibt, in dem rdy_in und eine Endbedingung $B_2$ erfüllt sind.

$$\text{AG}(\text{rdy_in} \wedge B_1 \rightarrow \text{AF}(\text{rdy_plc} \wedge F \wedge \text{AF}(\text{rdy_in} \wedge B_2)))$$

### Kategorie PRs4 - einfaches Verbot - Fremdbegrenzt

Analyseinhalt: Bei dieser Anforderungskategorie darf es im Beobachtungsintervall keinen Zustand geben, in dem die Folgerung erfüllt ist. Das Beobachtungsintervall beginnt mit einem Zustand, in dem die erste Bedingung $B_1$ erfüllt wird und endet mit einem Zustand, in dem die zweite Bedingung $B_2$ erfüllt wird. Dieses Verhalten lässt sich durch Erweiterung der Kategorie PRs5 erzielen, indem zusätzlich eine Endbedingung vereinbart wird.

$$\text{A}[\ (\ (\text{rdy_in} \wedge B_1) \rightarrow \text{AG}\ \neg(\text{rdy_plc} \wedge F\ )\ )\ \text{U}\ (\text{rdy_in} \wedge B_2)\ ]$$

### Kategorie POe4 - erweiterte Möglichkeit - Fremdbegrenzt

Analyseinhalt: Diese Kategorie lässt sich durch Verknüpfung der Kategorien POe5 und PRs5 erstellen. Eine bestimmte Folgerung darf nur zwischen zwei verschiedenen Bedingungen ausgeführt werden. Das Beobachtungsintervall wird durch die beiden Bedingungen $B_1$ und $B_2$ begrenzt. In der Umkehrung ergibt sich wiederum, dass die Folgerung nicht außerhalb des Beobachtungsintervalls erfüllt werden darf, d. h. weder vor der ersten Bedingung $B_1$ noch nach der zweiten Bedingung $B_2$.

$$\begin{aligned}&\text{AG}\ (\ (\ (\text{rdy_in} \wedge B_2) \rightarrow \neg\text{EF}\ (\text{rdy_plc} \wedge F\ )\ )\\&\quad \wedge \text{A}[\ \neg(\text{rdy_plc} \wedge F)\ \text{U}\ (\text{rdy_in} \wedge B_1)\ ]\ )\end{aligned}$$

### Kategorie DEe4 - erweiterte Forderung - Fremdbegrenzt

Analyseinhalt: Sobald es einen Zustand gibt, in dem rdy_in und eine Startbedingung $B_1$ erfüllt werden, muss es danach einen Zustand geben, in dem rdy_plc und eine Folgerung F erfüllt werden, bevor es einen Zustand gibt, in dem rdy_in und eine Endbedingung $B_2$ erfüllt sind. Zusätzlich darf es keinen Zustand mit rdy_plc und $F$ geben, bevor rdy_in und $B_1$ erstmals gültig waren und es darf auch keinen Zustand mit rdy_plc und $F$ geben darf, nachdem rdy_in und $B_2$ erstmals gültig waren.

$$\begin{aligned}&\text{AG}((\text{rdy_in} \wedge B_1 \rightarrow \text{AF}(\text{rdy_plc} \wedge F \wedge \text{AF}(\text{rdy_in} \wedge B_2)))\\&\quad \wedge (\text{A}[\text{C U } B_1] \vee \text{AG } C) \wedge \text{AG}(B_2 \rightarrow (\text{A}[C \text{ U } B_1] \vee \text{AG } C)))\end{aligned}$$

mit $C := \text{rdy_plc} \wedge \neg B_1 \wedge \neg F$

### Zeitparameter „Selbstbegrenzt“

#### Kategorie DEs3 - einfache Forderung - Selbstbegrenzt

Analyseinhalt: Sobald es einen Zustand gibt, in dem rdy_in und eine Bedingung *B* erfüllt werden, muss es danach einen Zustand geben, in dem sowohl rdy_plc, die Bedingung *B* und eine Folgerung *F* erfüllt werden, bevor die auslösende Bedingung *B* wieder ungültig wird. Die Forderung *F* findet also irgendwann innerhalb des Beobachtungsintervalls $B_0$ = (rdy_in $\wedge$ *B*) und $B_1$ = (rdy_in $\wedge \neg B$) statt. Forderungen dieser Form können in CTL nicht formuliert werden: *F* muss irgendwann für eine gewisse Zeitspanne nach dem Eintreten von $B_0$ gelten; ist diese Zeitspanne jedoch abgelaufen und gilt wiederum $\neg F$, gilt noch immer $B_0$, ohne dass gefordert wäre, dass *F* wiederum irgendwann Gültigkeit erlangt. Diese beiden Fälle sind in CTL jedoch nicht unterscheidbar.

Dieses Problem wurde durch eine Modifikation des Modells, d.h. des Registernetzes V*,* welches das System Anlage/Steuerung repräsentiert, gelöst [Hein01].

$$AG(r_0 = 1 \rightarrow A[r_1 = 0 \text{ U } (F \wedge AF\ r_1 = 1)]$$

$$\text{mit } r_0 := V, z \text{ sat } B_0 \text{ , } r_1 := V, z \text{ sat } B_1$$

#### Kategorie PRs3 - einfaches Verbot - Selbstbegrenzt

Analyseinhalt: Es wird gefordert, dass eine bestimmte Folgerung nicht ausgeführt werden darf, solange gleichzeitig eine bestimmte Bedingung erfüllt ist. Das Beobachtungsintervall umfasst also alle Zustände, in denen die Bedingung erfüllt ist. Dieser Umstand entspricht jedoch auch der Kategorie PRs1 - einfaches Verbot - Zustand. Die benutze Formel ist deshalb dieselbe.

$$AG\ (\ (\text{rdy_plc} \wedge B) \ \rightarrow \neg F\ )$$

bzw. wiederum:

$$AG\ \neg(\ \text{rdy_plc} \wedge B \wedge F\ )$$

### Kategorie POe3 - erweiterte Möglichkeit - Selbstbegrenzt

Analyseinhalt: Innerhalb dieser Anforderungskategorie wird gefordert, dass eine bestimmte Folgerung *F* nur dann ausgeführt werden darf, solange gleichzeitig auch eine festgelegte Bedingung erfüllt ist. Das Beobachtungsintervall wird also durch alle Zustände gebildet, in denen die Bedingung erfüllt ist. Bei Invertierung des Beobachtungsintervalls bedeutet dies, dass die bezeichnete Folgerung nicht ausgeführt werden darf, wenn die Bedingung nicht erfüllt ist. Dieses Verhalten kann auf die Kategorie POe1 (erweiterte Möglichkeit – Zustand) zurückgeführt werden.

$$\text{AG ( (rdy_plc} \wedge F) \rightarrow B \text{ ) bzw. AG ( (rdy_plc} \wedge \neg B) \rightarrow \neg F \text{ )}$$

bzw. nach Umformung:

$$\text{AG } \neg\text{( rdy_plc} \wedge \neg B \wedge F \text{ )}$$

### Kategorie DEe3 - erweiterte Forderung - Selbstbegrenzt

Analyseinhalt: Sobald es einen Zustand gibt, in dem rdy_in und eine Bedingung *B* erfüllt werden, muss es danach einen Zustand geben, in dem sowohl rdy_plc, die Bedingung *B* und eine Folgerung *F* erfüllt werden, bevor die auslösende Bedingung *B* wieder ungültig wird. Zusätzlich darf in allen Zuständen, in denen zwar rdy_plc jedoch nicht die Bedingung *B* erfüllt wird, die Folgerung *F* auch nicht erfüllt sein. Diese Kategorie lässt sich wiederum durch Verknüpfung zweier anderer Kategorien, nämlich DEs3 und POe3, erstellen.

$$\text{AG}((r_0 = 1 \rightarrow \text{A}[r_1 = 0 \text{ U } (F \wedge \text{AF } r_1 = 1)]$$
$$\wedge (\text{A}[C \text{ U } r_0 = 1] \vee \text{AG } C) \wedge \text{AG}(r_1 = 1 \rightarrow (\text{A}[C \text{ U } r_0 = 1]) \vee \text{AG } C)))$$

$$\text{mit } C := \text{rdy_plc} \wedge \neg B \wedge \neg F$$

Ergebnis der Übersetzung der mit der Sicherheitsfachsprache formulierten Anforderungen ist eine Menge temporallogischer Formeln. Die atomaren Präpositionen dieser Formeln korrespondieren mit Variablen des Systemmodells, also letztendlich auch wiederum mit den Einträgen im Variablendeklarationsteil des SPS-Programms.

# 4 Prototypische Realisierung und Einsatz der Sicherheitsfachsprache

In den vorherigen Abschnitten wurde deutlich gemacht, welche typischen steuerungstechnischen Problemstellungen es gibt und wie diese mit der Sicherheitsfachsprache dargestellt werden können. In diesem Abschnitt wird nun dargelegt, auf welche Art und Weise und mit welchen Hilfsmitteln diese Anforderungen niedergeschrieben und in temporallogische Formeln umgewandelt werden können.

## 4.1 Technische Umsetzung, Aufbaustruktur

Die im Zusammenhang mit der SFS erstellten Software-Tools unterteilen sich in zwei Bereiche:

- SFS-Editor, ein syntaxgesteuerter Editor zur Erstellung SFS-konformer Anforderungssätze,
- Compilersequenz, die aus den natürlichsprachlichen Anforderungen CTL-Formeln generiert.

Mit dieser strikten Trennung (siehe Abbildung 18) zwischen erstellenden und überführenden Softwarekomponenten ist eine einfache Austauschbarkeit und Anpassbarkeit der einzelnen Module an sich ändernde Anforderungen gegeben.

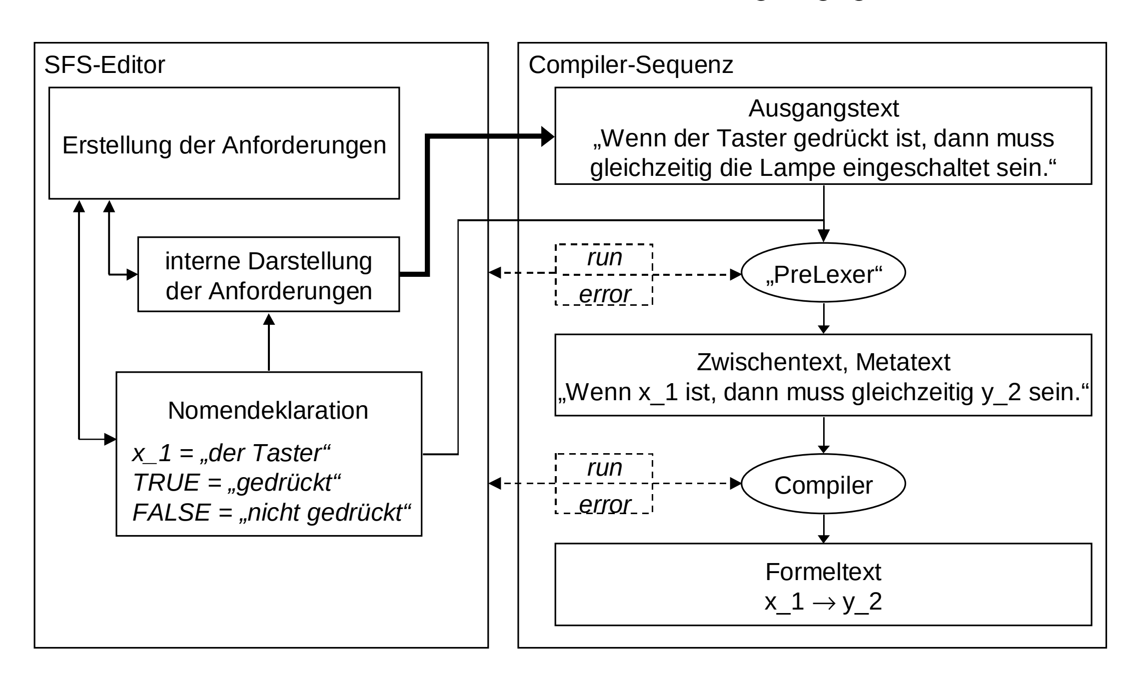



*Abbildung 18 - Struktur des Prototypen der Sicherheitsfachsprache*

Als Datenschnittstellen zwischen den einzelnen Modulen dienen ASCII-Textdateien, die auch unabhängig von den einzelnen Modulen erstellt und ausgewertet werden können. Dies sind:

- Nomendeklaration, diese enthält die Zuordnung zwischen den SPS-internen Variablenbezeichnungen und den natürlichsprachlichen Ausdrücken,
- Ausgangstext, dieser enthält die syntaktisch korrekten, vollständigen natürlichsprachlichen Anforderungen,
- Zwischentext (Metatext), dieser enthält den bereits syntaktisch analysierten und überführten Metatext,
- Formeltext, dieser enthält die in CTL-Formeln dargestellten Programmanforderungen.

Prinzipiell können diese Dateien (vielleicht abgesehen vom Formeltext) separat per Hand erstellt werden und einzeln und unabhängig in die Überführungssequenz eingespeist werden. Wegen der Komplexität und Fehleranfälligkeit dieses Vorgangs bietet sich jedoch die Verwendung des nutzerfreundlichen SFS-Editors an.

Die folgenden Abschnitte beschreiben in kurzer Form die einzelnen Softwaremodule im Umfeld der Sicherheitsfachsprache und erläutern darüber hinaus die Vorgehensweise bei der Erstellung von Anforderungen mit der SFS. Dabei werden zunächst die Erstellung der verbalen Anforderungen beschrieben, bevor auf die überführenden Module (d. h. den PreLexer und den Compiler) eingegangen wird. Dabei wird auch dargestellt, in welcher Form die Schnittstellen zwischen den einzelnen Modulen beschaffen sein müssen.

## 4.2 Erstellung von Programmanforderungen mit dem SFS-Editor

Mit Hilfe des syntaxgesteuerten Editors für die Sicherheitsfachsprache ist es auf einfache Weise möglich, SFS-konforme Anforderungssätze zu erstellen. Natürlich können diese Anforderungssätze prinzipiell vom Anwender auch per Hand mit einem Texteditor erzeugt und anschließend kompiliert werden, jedoch sind hier der Aufwand und die Fehleranfälligkeit weitaus größer.

Die Erstellung der Anforderungssätze erfolgt in mehreren Stufen, die in einer Vorbereitungsphase und einer Durchführungsphase zusammengefasst werden können (siehe Abbildung 19).

In der Vorbereitungsphase ist zunächst die Verfügbarkeit der benötigten *Verbalphrasen* zu überprüfen. Hier unterscheidet sich die Anwendung des SFS-Editors innerhalb einer Verifikationsumgebung von der Benutzung innerhalb eines Syntheseprozesses. Während bei der Verifikation eines bereits bestehenden Steuerungsprogramms alle SPS-Variablen festgelegt sind und aus dem Variablendeklarationsteil des Programms ausgelesen werden können, sind diese Variablen bei der Erstellung eines neuen Programms noch nicht bekannt und müssen erst definiert werden. Der SFS-Editor ist für die Nutzung in beiden Umgebungen geeignet: SPS-Variablen können sowohl aus bereits vorhandenen Programmen eingelesen als auch neu definiert und editiert werden. Ziel der Vorbereitungsphase ist es, für alle benötigten SPS-Vari-

ablen natürlichsprachliche, applikationsspezifische *Verbalphrasen* (Nomen und Werte) zu definieren.

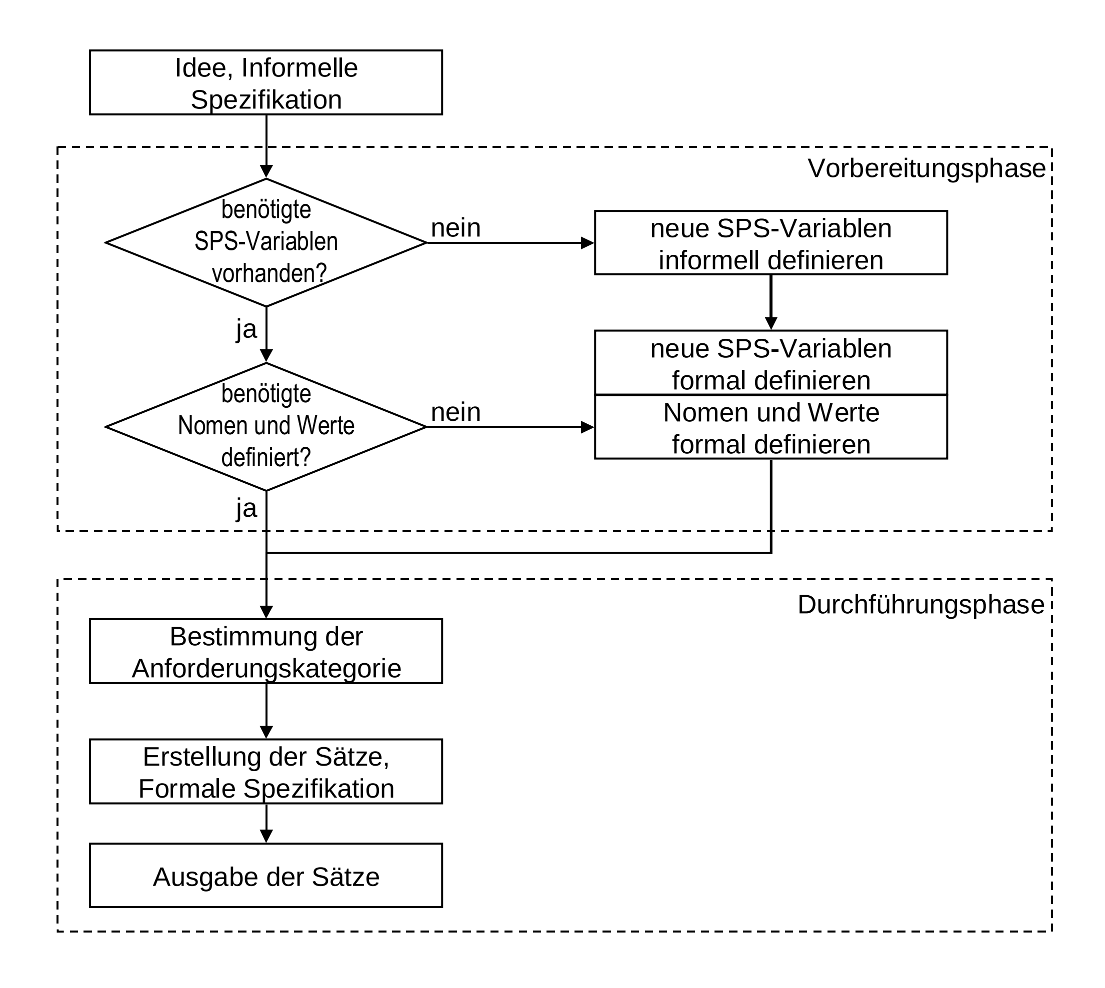



*Abbildung 19 - Ablauf der Erstellung von Anforderungen mit dem SFS-Editor*

In der Durchführungsphase bestimmt der Nutzer zunächst die Anforderungskategorie, die seiner Idee von einer speziellen Programmanforderung entspricht. Damit ist ein Anforderungskonstrukt festgelegt, das mit den zuvor erstellten Verbalphrasen ausgefüllt wird.

In der Abbildung 20 ist die Erstellung der Verbalphrasen mit dem SFS-Editor dargestellt. Die in der Fallstudie verwendeten SPS-Variablen können dem Variablendeklarationsteil des zu analysierenden SPS-Programms entnommen werden. Anschließend ist für jede Variable und für jeden Wert, den diese annehmen kann, eine verbale Interpretation zu erstellen. Die Auflistung der Variablen und ihre verbale Interpretation wird in der Nomendeklarationsdatei abgelegt und in der Compiler-Sequenz wieder verwendet. Sollten zunächst nicht alle Variablen bekannt sein, kann das Erstellen der Nomen und Werte auch zu einem späteren Zeitpunkt erfolgen.

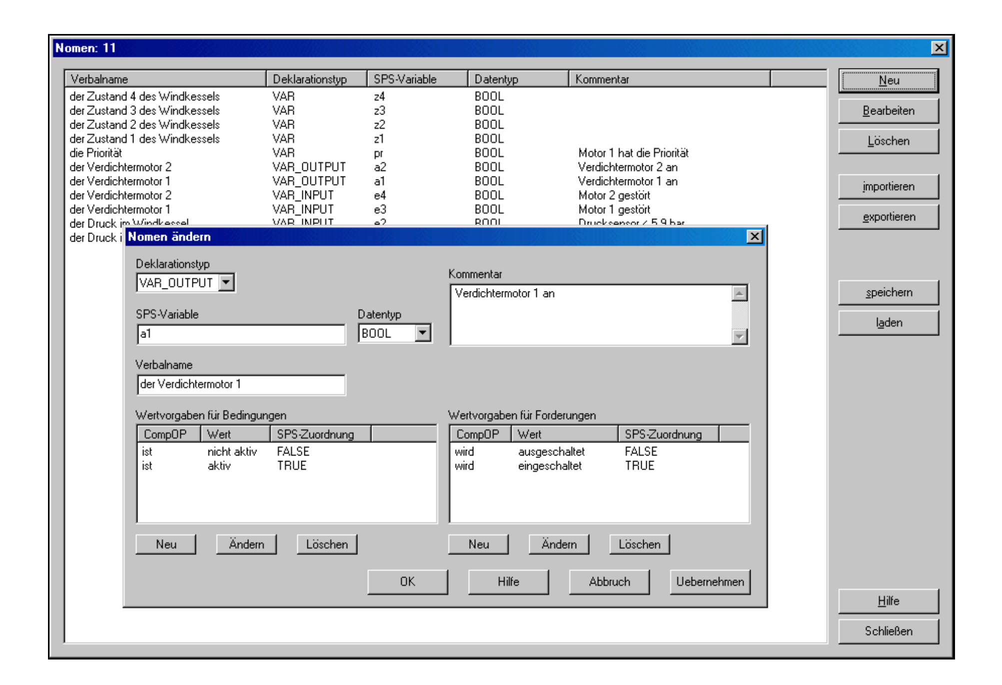



*Abbildung 20 - Erstellung der Verbalphrasen*

Nachdem alle SPS-Variablen interpretiert worden sind, können die eigentlichen Anforderungssätze erstellt werden. Dazu wird im SFS-Editor ein neuer Satz erzeugt und es werden der Modalparameter und der Zeitparameter bestimmt (Abbildung 21).

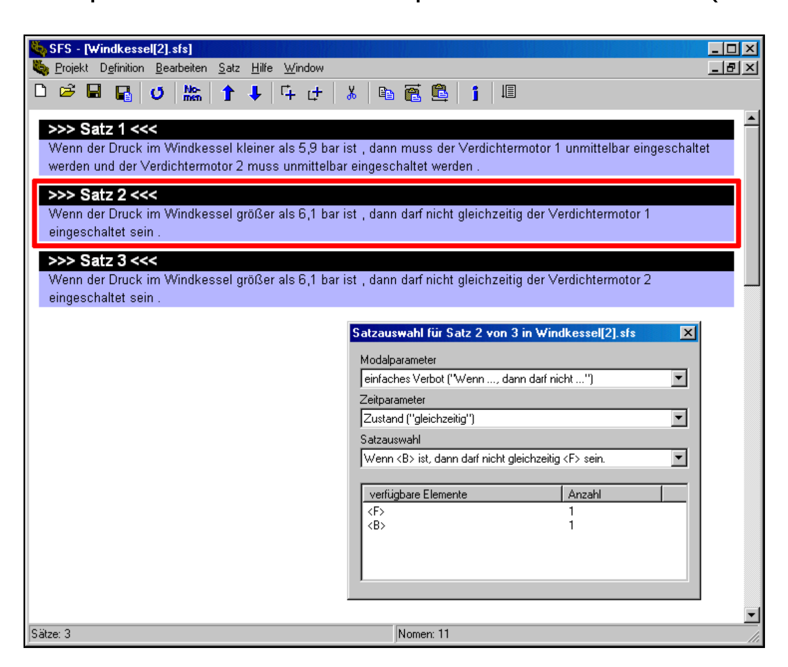



*Abbildung 21 - Erstellung der Anforderungssätze*

Durch die Bestimmung des Modal- und Zeitparameters ist der Anforderungsinhalt prinzipiell bestimmt, durch eine weitere Auswahl kann die Struktur des Satzes bestimmt werden. An dieser Stelle werden dem Anwender mehrere inhaltlich redundante Satzmuster angeboten, die er nach seinem Geschmack auswählen kann. Diese inhaltliche Redundanz bezieht sich vor allem auf die Stellung der Bedingung und der Folgerung im Satz und auf die Auswahl verschiedener gleichwertiger Schlüsselwörter.

Anschließend ist der Anforderungssatz durch Erstellung der Bedingung/-en und der Folgerung/-en zu vervollständigen (Abbildung 22). Auch hier werden wieder gleichwertige Formulierungen angeboten, die sich jedoch nur durch die Stellung der einzelnen Wörter innerhalb der Formulierung unterscheiden.

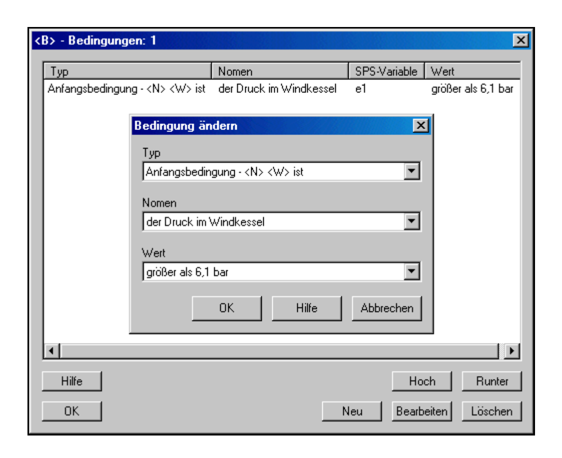



*Abbildung 22 - Erstellung der Bedingungen*

An dieser Stelle können auch immer mehrere Bedingungen und Folgerungen vereinbart werden, diese sind in der Anforderung dann jeweils konjunktiv miteinander verknüpft.

In einem letzten Schritt werden die erstellten Sätze in eine Textdatei ausgegeben und stehen nun für die Kompilierung in eine CTL-Formel zur Verfügung.

## 4.3 Überführung der Programmanforderungen in CTL-Formeln

Die natürlichsprachlichen Programmanforderungen, die im so genannten Ausgangstext abgelegt sind, werden durch eine Compilersequenz in CTL-Formeln überführt. Dieser Umwandlungsvorgang besteht aus zwei Stufen:

1. Im ursprünglichen Text ersetzt ein „PreLexer“ die applikationsspezifischen Wortphrasen durch die in der Nomendeklarationsdatei hinterlegten SPS-Variablen und deren Werte. Der dadurch entstandene Zwischen-(Meta-)text enthält dann nur noch sprachinterne, applikationsunabhängige Schlüsselwörter sowie formale Ausdrücke des SPS-Programms.

2. Im eigentlichen Umwandlungsvorgang wird der Zwischentext hinsichtlich der Bedeutung analysiert und in CTL-Formeln umgewandelt.

### *4.3.1 PreLexer*

Der PreLexer ist das erste Modul innerhalb der Compilersequenz. Als Ausgangsinformationen liegen dem PreLexer die natürlichsprachlichen Anforderungen sowie die Nomendeklaration in Form von zwei ASCII-Dateien vor. Aufgabe des PreLexers ist es, die rein verbalen Anforderungen für den nachfolgenden Kompiliervorgang vorzubereiten und zu vereinfachen. Hierbei werden die logisch zusammengehörigen Nomen und Werte innerhalb des Ausgangstextes wieder zu ***Nomen-Wert-Paaren*** zu-

sammengefasst und in die SPS-internen Variablendarstellungen überführt (z. B. *„die Presse“* und *„gestoppt“* zu ¬x_2). Im Gegensatz zum eigentlichen Compiler „kennt“ der PreLexer weder die Bedeutung, noch die Reichweite einer Anforderung. Er orientiert sich lediglich am Satzbau bzw. der Positionierung der in der Nomendeklarationsdatei abgelegten Wortphrasen sowie SFS-typischer Schlüsselwörter.

Die interne Umsetzung erfolgt dabei in zwei Schritten:

1. Umstellen einzelner Wörter im Satz, so dass danach das Nomen und der Wert nebeneinander und im richtigen Verhältnis zu den anderen Schlüsselwörtern stehen.
2. Zusammenfassen des Nomens und des Wertes und Ersetzen durch die SPS-Variable mit dem entsprechenden Wert.

Aus dem Text

*/* Satz 1 */*

*Wenn „der Zustand 1 der Presse“ „aktiv“ ist und „die Presse“ ist „unten“ , dann muss „das Schliessen der Presse“ unmittelbar „gestoppt“ werden und „das Öffnen der Presse“ muss unmittelbar „gestartet“ werden und „der Zustand 1 der Presse“ muss unmittelbar „zurückgesetzt“ werden und „der Zustand 2 der Presse“ muss unmittelbar „gesetzt“ werden .*

*/* Satz 2 */*

*Wenn „der Zustand 2 der Presse“ „aktiv“ ist und „die Presse“ ist „oben“ , dann muss „das Öffnen der Presse“ unmittelbar „gestoppt“ werden und „der Zustand 2 der Presse“ muss unmittelbar „zurückgesetzt“ werden und „der Zustand 3 der Presse“ muss unmittelbar „gesetzt“ werden .*

*/* Satz 3 */*

*Wenn „die Presse“ „unten“ ist , dann darf „das Ausfahren des Roboterarmes 1“ nicht gleichzeitig „gestartet“ sein .*

wird durch den PreLexer der Metatext

```
/* Satz 1 */
WENN press_z1 IST UND press_bottom IST, DANN MUSS UNMITTELBAR
!press_down WERDEN UND MUSS UNMITTELBAR press_upward WERDEN
UND MUSS UNMITTELBAR !press_z1 WERDEN UND MUSS
UNMITTELBAR press_z2 WERDEN .

/* Satz 2 */
WENN press_z2 IST UND press_top IST, DANN MUSS UNMITTELBAR
!press_upward WERDEN UND MUSS UNMITTELBAR !press_z2 WERDEN
UND MUSS UNMITTELBAR press_z3 WERDEN .

/* Satz 3 */
WENN press_bottom IST, DANN DARF NICHT GLEICHZEITIG arm1_forward
SEIN .
```

erzeugt. Dieser Text enthält nur noch SFS-typische Schlüsselwörter sowie SPS-Variablen.

### 4.3.2 Compiler

Während die Arbeitsweise des PreLexers in einer reinen Texterkennung, -umwandlung und -umstellung besteht, erfordert die Programmierung des Compilers weitreichendere Analysefunktionen. Aus diesem Grund wurde der Compiler mit den beiden Werkzeugen „flex“ und „bison“ (Abarten der bekannten Compilergeneratoren „lex“ und „yacc“) erstellt, die eine zeit- und kostengünstige Erstellung des Compilers, aber auch eine leichte Anpassbarkeit an zukünftige Erfordernisse gewährleisten.

Ein weiterer Vorteil dieser Vorgehensweise ist, dass die im Abschnitt 3.1 definierte und im Anhang A.2 dargestellte Struktur der Sicherheitsfachsprache direkt als Input für die Compilergeneratoren dienen kann.

Der Compiler generiert automatisch aus dem Zwischentext des PreLexers die CTL-Formeln:

```
/* Satz 1 */
AG ( (rdy_in & ( press_z1 & press_bottom ) ) -> A[
!rdy_plc U (rdy_plc & ( !press_down & press_upward &
!press_z1 & press_z2 ) ) ] )

/* Satz 2 */
AG ( (rdy_in & ( press_z2 & press_top ) ) -> A[ !rdy_plc U
(rdy_plc & ( !press_upward & !press_z2 & press_z3 ) ) ] )

/* Satz 3 */
AG !( rdy_plc & press_bottom & arm1_forward)
```

Diese CTL-Formeln können anschließend beispielsweise durch einen Model-Checker an einem Kontrollmodell des Steuerungsprogramms verifiziert oder für andere Zwecke verwendet werden.

## 4.4 Bildung von Verbalphrasen durch Interpretation der SPS-Variablen

Bei der Formulierung einer Anforderung mit der Sicherheitsfachsprache wird durch die Auswahl des Modal- und Zeitparameters ein bestimmtes Satzmuster festgelegt (siehe Anhang A.2 und A.3). Die noch bestehenden Freiräume innerhalb des Bedingungs- und des Folgerungsteils dieses Satzmusters müssen anschließend mit applikationsspezifischen Verbalphrasen ausgefüllt werden. Bei Betrachtung der Grundstruktur einer beliebigen Anforderungskategorie besteht diese zunächst aus dem konditionalen Grundmuster (Konjunktion aus zwei Teilsätzen) und verschiedenen allgemeinen Schlüsselwörtern (Zeitadverbien, Modalverben). Zur Vervollständigung des Satzes fehlen - gemäß den Grammatikregeln der deutschen Sprache - weitere typische Elemente, wie z. B. Subjekte für die Teilsätze und die zugehörigen Prädikate.

*„Wenn ... ist, dann muss gleichzeitig ... sein."*

Diese fehlenden Satzbausteine müssen durch den Benutzer der Sicherheitsfachsprache durch Interpretation der gegebenen SPS-Variablen und deren relevanten Werten gebildet werden. Der Variablendeklarationsteil des gegebenen SPS-Programms mit den dort festgehaltenen Informationen über die vorhandenen Aktor- und Sensorsignale sowie die internen Variablen bildet die Grundlage für die Erstellung der Verbalphrasen.

Eine Verbalphrase besteht immer aus zwei Teilen: einem ***Nomen*** und einem ***Wert***. Man kann eine Verbalphrase deshalb auch als eine Nomen-Wert-Kombination bezeichnen. Das Nomen wird dabei durch Interpretation einer SPS-Variablen, der Wert durch Interpretation eines bestimmten Wertes, den diese SPS-Variable annehmen kann, gebildet. Da SPS-Variablen typischerweise mehrere Werte annehmen können, gibt es zu einem Nomen immer auch mehrere Werte und somit auch immer mehrere Nomen-Wert-Kombinationen zu einer SPS-Variablen. So kann z. B. zu der booleschen SPS-Variablen „`x_1`", die einen bestimmten Taster repräsentiert, das *Nomen* *„der Start-Taster"* erstellt werden, für den „TRUE"-Zustand kann der *Wert* *„gedrückt"* und der „FALSE"-Zustand der *Wert* *„nicht gedrückt"* vereinbart werden. Für diese Variable ergeben sich die beiden Nomen-Wert-Kombinationen:

`x_1` - „*der Start-Taster*" ist „*gedrückt*"
¬ `x_1` - „*der Start-Taster*" ist „*nicht gedrückt*"

Diese Zuordnung ist für alle verwendeten Variablen einmal zu erstellen. Bei der anschließenden Formulierung der Anforderungen werden nur noch diese erzeugten *Nomen* und *Werte* benutzt, ohne dass sich der Anwender ständig über die dahinter liegenden steuerungstechnischen Gegebenheiten im Klaren sein muss. Bei der Vervollständigung der Anforderungssätze werden die Nomen als Subjekte innerhalb der Teilsätze verwendet, die *Werte* bilden zusammen mit den bereits vorhandenen Hilfsverben (*„ist"*, *„sein"*, *„werden"*) die zugehörigen Prädikate. Als *Wert* und damit auch als Prädikat können dabei nicht nur einzelne Wörter dienen, hier sind - je nach Problemstellung und Initiative des Anwenders - auch größere Zusammensetzungen möglich: Verbindungen von weiteren Verben, Substantiven und Adjektiven, Adverbien, Umstandsbestimmungen, Vergleichsoperatoren usw.

Die Erstellung der Nomen und Werte ist ein Prozess, der eine gewisse Kreativität und Weitsicht des Anwenders verlangt. Hierbei sind verschiedene Randbedingungen zu beachten, damit einerseits die bestehenden Restriktionen der SFS eingehalten werden, andererseits eine möglichst freie und natürliche Formulierung möglich ist. In den folgenden Absätzen werden einige Informationen und Anregungen gegeben, wie die erforderlichen Nomen und Werte zu erstellen sind.

### Einsatz der Nomen-Wert-Kombinationen in Bedingungen bzw. Folgerungen

Die erzeugten Nomen-Wert-Kombinationen werden sowohl bei der Formulierung des Bedingungs- als auch des Folgerungsteils der Anforderungssätze verwendet. Als Besonderheit ist zu erwähnen, dass SPS-Variablen, die Sensoren repräsentieren nur innerhalb von Bedingungen sinnvoll verwendet werden können. Aktor-Variablen werden dagegen typischerweise für die Darstellung der Folgerungen verwendet, können aber auch in Bedingungen eingesetzt werden. SPS-interne Variablen werden in Bedingungen und in Folgerungen gleichermaßen verwendet. Es erweist sich aus diesem Grund als sinnvoll und auch notwendig, für Aktor- und für interne Variablen zusätzliche Werte zu definieren, um den veränderten Gegebenheiten bei der Benutzung in einer Bedingung oder einer Folgerung Rechnung zu tragen. Bei Bedingungen wird der Wert einer Variablen abgefragt bzw. mit einem anderen Wert verglichen, bei Folgerungen wird einer Variablen ein neuer Wert zugewiesen bzw. es wird die Zuweisung eines Wertes verboten.

| | | |
|---|---|---|
| y_1 | - bei Verwendung in einer Bedingung | „*der Motor*“ ist „*aktiv*“ |
| | - bei Verwendung in einer Folgerung | „*der Motor*“ wird „*gestartet*“ |
| ¬ y_1 | - bei Verwendung in einer Bedingung | „der Motor“ ist „nicht aktiv“ |
| | - bei Verwendung in einer Folgerung | „*der Motor*“ wird „*gestoppt*“ |

Durch die unterschiedliche Verwendung des Hilfsverbs „sein“ in den Bedingungen und Folgerungen (einerseits in einem passiven Zusammenhang, andererseits in einem aktiven Zusammenhang) ergibt sich die Notwendigkeit einer unterschiedlichen Definition der Werte.

### Formulierung der Nomen-Wert-Kombinationen für Bool- bzw. Integer-Variablen

Während Variablen des Datentyps Bool nur zwei Zustände aufweisen und somit die Anzahl der formulierbaren Werte begrenzt ist (siehe oben), besteht diese Begrenzung für Variablen des Datentyps Integer nur aufgrund des darstellbaren Wertebereichs. Jedoch ist es auf der anderen Seite nicht möglich und auch nicht notwendig, den gesamten Wertebereich einer Integer-Variablen vollständig „verbalisieren“ zu wollen. Betrachtet man sich die Art und Weise der Verwendung von Integer-Variablen in SPS-Programmen, so fallen einige wenige typische Anwendungsfälle auf: (ohne dass die folgende Liste vollständig ist)

- intern als Zustandsvariable zur Kodierung von Programmzuständen,
- externe Sensorwerte,
- externe Aktorwerte,

Die Anzahl der unterscheidbaren Zustände eines Programms ist begrenzt, so dass hier eine Erstellung der Werte für jeden der einnehmbaren Variablenzustände möglich ist.

| | | |
|---|---|---|
| z_1 = 1 | - als Bedingung | „*das Programm*“ ist „*im Zustand 1*“ |
| | - als Folgerung | „das Programm“ wird „in den Zustand 1 überführt“ |
| z_1 = 2 | - als Bedingung | „das Programm“ ist „im Zustand 2“ |
| | - als Folgerung | „das Programm“ wird „in den Zustand 2 überführt“ |
| ... | ... | ... |

In ereignisorientierten Steuerungsprogrammen, in denen Integer-Variablen zum Einlesen analoger Sensorwerte, bzw. zum Ausgeben analoger Aktorwerte verwendet werden, erfolgt in vielen Fällen eine Intervallbildung bzw. Diskretisierung des verfügbaren Wertebereichs. Bei analogen Sensoren werden bestimmte Grenzwerte abgefragt, bei deren Über- oder Unterschreitung Aktionen auszulösen sind. Auf diese Weise lassen sich relevante Werte dieser Variablen identifizieren, für die verbale Entsprechungen zu formulieren sind. In diesem Zusammenhang ist auch die Benutzung von Vergleichsoperatoren möglich.

x_t = 50 - „die Temperatur" ist „gleich 50°C"
x_t > 50 - „die Temperatur" ist „größer als 50°C"
x_t < 50 - „die Temperatur" ist „kleiner als 50°C"

**Sichtenorientierte Formulierung der Nomen-Wert-Kombinationen**

Ein weiterer Punkt der kreativen Einflussnahme des Anwenders der Sicherheitsfachsprache auf die Lesbarkeit und Verständlichkeit seiner Anforderungssätze ergibt sich durch einen unterschiedlichen Abstraktionsgrad bei der Erstellung der Nomen und Werte. Möglich sind hier:

- eine vorwiegend variablenorientierte Beschreibung,
- eine prozessorientierte Beschreibung.

Bei einer variablenorientierten Beschreibung erfolgt lediglich eine direkte, nichtinterpretierte Überführung der SPS-Variablen und ihrer Werte in die Nomen und Werte.

x_1 - *„die Variable x_1"* ist *„TRUE"* (alternativ: „wahr", „gesetzt")
¬ x_1 - *„die Variable x_1"* ist *„FALSE"* (alternativ: *„falsch", „nicht gesetzt"*)

Mit dieser Darstellung kann man prinzipiell bereits umgehen, die entscheidenden Möglichkeiten der Sicherheitsfachsprache bestehen jedoch vor allem auch darin, die verwendeten SPS-Termini prozessorientiert zu interpretieren und somit eine bessere Verständlichkeit und auch eine Erhöhung der Anwendungssicherheit zu erzielen. Hierbei kann der Anwender genau die technische Bedeutung der einzelnen Variablenzustände angeben und somit Missverständnisse und Fehlinterpretationen vermeiden.

| | | |
|---|---|---|
| x_2 | - *„die Variable x_2" ist „gesetzt"* | (variablenorientiert) |
| | - *„der Not-Aus-Taster" ist „nicht gedrückt"* | (prozessorientiert) |
| ¬ x_2 | - *„die Variable x_2" ist „nicht gesetzt"* | (variablenorientiert) |
| | - *„der Not-Aus-Taster" ist „gedrückt"* | (prozessorientiert) |

Der Grad der Interpretation kann natürlich, je nach Notwendigkeit, angepasst werden:

x_3 - *„die Variable x_3" ist „gesetzt"* (variablenorientiert)
- *„der Endlagenschalter x_3*" ist *„gedrückt*" (prozessorientiert)
- „der Werkzeugschlitten" ist „in der linken Endstellung"

Die Abbildung 23 fasst die verschiedenen Möglichkeiten der Erstellung der Verbalphrasen noch einmal zusammen.

| Einsatz der Variablen | Datentyp | Sensoren | Aktoren | int. Variablen |
|---|---|---|---|---|
| ... in einer Bedingung<br><br>*„Wenn ... "* | Bool | **Abfrage eines Sensorzustandes**<br>**x_1 = TRUE**<br>*„der Hydraulikzylinder*" ist *„oben*"<br>*„der Kran*" ist *„in Entnahmeposition*"<br>*„die Lichtschranke*" ist *„blockiert*" | **Abfrage einer laufenden Aktion**<br>**y_1 = TRUE**<br>*„das Öffnen der Presse*" ist *„aktiv*"<br>*„der Magnet*" ist *„aktiv*" | **Abfrage eines besteh. Zustandes**<br>**z_1 = TRUE**<br>*„der Zustand 1 der Presse*" ist *„aktiv*" |
| | Integer | **Abfrage eines Sensorzustandes**<br>**x_2 = 50**<br>*„die Temperatur*" ist *„50° Celsius*"<br>*„der Roboter*" ist *„vor der Presse*" | **Abfrage einer laufenden Aktion**<br>**y_2 = 50**<br>*„die Drehzahl des Motors*" ist *„50 UpM*" | **Abfrage eines besteh. Zustandes**<br>**z_2 = 50**<br>*„der Wert des Merkers*" ist *„50*" |
| ... in einer Folgerung<br><br>*„dann ... "* | Bool | | **Start oder Ende einer Aktion**<br>**y_1 = TRUE**<br>*„die Presse*" wird *„geschlossen*"<br>*„der Magnet*" wird *„aktiviert*" | **Übergang in einen neuen Zustand**<br>**z_1 = TRUE**<br>*„der Zustand 1 der Presse*" wird *„gesetzt*" |
| | Integer | | **Veränderung einer Aktion**<br>**y_2 = 50**<br>*„die Drehzahl des Motors*" wird *„auf 50 UpM gesetzt*" | **Übergang in einen neuen Zustand**<br>**z_2 = 50**<br>*„der Merker*" wird *„auf 50 gesetzt*" |

*Abbildung 23 - Möglichkeiten zur Erstellung der Verbalphrasen*

Eine letzte Anmerkung soll den grammatikalischen Möglichkeiten der Sicherheitsfachsprache gelten. Bei Anforderungen, deren Bedingungs- bzw. Folgerungsteil mehrere konjunktiv verknüpfte Teilbedingungen bzw. Teilfolgerungen enthält, kann (gemäß den Gegebenheiten der deutschen Umgangssprache) eine Umstellung der Satzglieder erfolgen.

Wenn *„die Variable 1*" ist *„gesetzt*" und *„die Variable 2*" ist *„gesetzt*", dann ...

Wenn *„die Variable 1*" *„gesetzt*" ist und *„die Variable 2*" ist *„gesetzt*", dann ...
Wenn *„die Variable 1*" *„gesetzt*" ist und *„die Variable 2*" *„gesetzt*" ist, dann ...

Im vorliegenden Fall wurde die Reihenfolge des Hilfsverbs „ist" und des Wertes *„gesetzt*" vertauscht, um die geforderte Wortreihenfolge einzuhalten. Die Sicherheitsfachsprache ist auf solche grammatikalischen Besonderheiten ausgelegt, der Anwender kann die ihm passend erscheinende Wortfolge auswählen. Bei der Überfüh-

rung der Anforderungssätze in die temporallogischen Formeln werden diese Variationsmöglichkeiten ebenfalls berücksichtigt.

## 4.5 Nutzung der Sicherheitsfachsprache für Verifikation oder Synthese

Neben der inhaltlichen Kategorisierung der Anforderungen durch den Modal- und den Zeitparameter lassen sich die Anforderungen an ein Steuerungsprogramm auch hinsichtlich ihrer Erzeugbarkeit in Abhängigkeit der Existenz eines realen Steuerungsprogramms einteilen. So sind die beiden folgenden Kategorien unterscheidbar:

- **Realisierungsunabhängige Anforderungen**: diese beziehen sich ausschließlich auf die generell bekannten Variablen (Sensoren, Aktoren) und sind somit unabhängig von einem untersuchten SPS-Programm anwendbar.
- **Realisierungsabhängige Anforderungen**: diese beziehen sich neben den verwendeten Sensoren und Aktoren auch auf die internen Variablen des untersuchten SPS-Programms und sind somit auch nur auf dieses anwendbar.

Diese Einteilung wird besonders dann deutlich, wenn man sich die unterschiedlichen Anforderungen bei der Spezifikation eines Steuerungsprogramms für eine Verifikation bzw. für eine Synthese des Steuerungsprogramms betrachtet. Im ersten Fall existiert das zu untersuchende Steuerungsprogramm bereits, d. h. alle für eine Formulierung der Anforderungen in Frage kommenden Variablen sind bereits definiert, und müssen dementsprechend verwendet werden. Dagegen sind bei einer Spezifikation für eine nachfolgende Programmsynthese lediglich die verfügbaren Sensor- und Aktor-Variablen bekannt, weitere programminterne Variablen (wie Zustands- oder Hilfsvariablen) sind noch nicht bekannt und müssen definiert werden.

Realisierungsunabhängige Anforderungen sind somit umfassender einsetzbar, realisierungsabhängige Anforderungen sind dagegen nur auf bereits existierende Steuerungsprogramme anwendbar.

## 4.6 Systematische Erstellung von Sicherheitseigenschaften

Als ein besonderes Problem bei der Erstellung von allgemeingültigen, (realisierungsunabhängigen) Anforderungen an ein Steuerungsprogramm erweist sich die Zusammenstellung von Situationen, Zuständen bzw. Programmreaktionen, die niemals erreicht bzw. ausgeführt werden dürfen, weil sie eine Gefährdung des beteiligten Personals, der Umwelt, der Anlage oder des eingesetzten Materials darstellen. Der Steuerungstechniker sieht sich hierbei, auch in Kooperation mit verschiedenen anderen Projektbeteiligten, meist nicht in der Lage, alle denkbaren Gefährdungssituationen zu erkennen, zu beurteilen und zu verhindern. Solche Gefährdungssituationen werden oft erst bei bzw. nach einem Unglück erkannt, die notwendigen steuerungstechnischen Gegenmaßnahmen können erst anschließend in das Programm integriert werden. Demnach ist es außerordentlich wichtig, besonders sorgfältig und möglichst umfassend bei der Erstellung von Sicherheitseigenschaften vorzugehen. Erkannte Fehler und Gefahrensituationen auslösende Bedingungen dürfen nach ei-

ner Überarbeitung des Programms nicht ein zweites Mal auftreten. Aus diesem Grund ist es sinnvoll, mit geeigneten Verfahren und Methoden das betrachtete System zielgerichtet zu analysieren, um eventuelle Risiken aufzudecken.

In diesem Abschnitt werden einige Verfahren gezeigt, mit deren Hilfe es möglich ist, denkbare Störungen und Risiken, sowie die zu dem unerwünschten Ereignis führenden Ursachen, zu identifizieren. Die Vielzahl der derzeit verfügbaren Verfahren lässt sich, je nach Herangehensweise an das Problem, in induktive und deduktive Verfahren unterteilen.

Bei den *induktiven Verfahren* werden, ausgehend vom Ausfall eines Elementes oder dem Eintreten einer bestimmten Bedingung, die Ereignisse ermittelt, die dieser Ausfall bzw. diese Bedingung hervorrufen kann.

Beispiele hierfür sind:

- Vorläufige Gefahrenquellenanalyse (engl. preliminary hazard analysis, PHA)
- Ausfall- und Wirkungsanalyse (engl. failure mode and effect analysis, FMEA) nach DIN 25448
- Ereignis- (Störablauf-) -analyse (engl. event tree analysis, ETA) nach DIN 25419

Bei den *deduktiven Verfahren* wird der umgekehrte Weg gegangen: ausgehend von dem zu vermeidenden Schadensfall werden die möglichen Ursachen und Wege dahin analysiert.

Beispiel:

- Fehlerbaumanalyse (engl. Fault tree analysis, FTA; Cause tree method, CTM) nach DIN 25424

Wie zu erkennen ist, sind die meisten dieser Verfahren bereits standardisiert, darüber hinaus gibt es auch intensive Forschungen auf diesem Gebiet. In [Hube97] wird ein Verfahren der qualitativen Systemanalyse beschrieben (HAZOP), das eine computerunterstützte Gefahrenidentifikation im Bereich der Chemietechnik ermöglicht. Hierbei werden gesicherte Wertebereiche der jeweiligen technischen Größen festgelegt, mit denen anschließend so genannte Situationstabellen berechnet werden, die anzeigen sollen, ob eine gefährliche Situation eintreten kann oder nicht.

Ein ähnliches Verfahren der Gefahrenanalyse wird in [Bieg98] dargestellt. Das Verfahren SQMA beinhaltet eine simulationsbasierte qualitative Modellbildung und Analyse des untersuchten Systems, wobei alle technischen Möglichkeiten, die das System einnehmen kann, aufgestellt werden. An dieser Aufstellung lassen sich anschließend die gefährlichen Zustände identifizieren.

Ein besonderer Schwerpunkt der Aktivitäten erwächst aus dem Bereich der Luftfahrt- und Automobilindustrie [Stöl98]. Gerade der Einsatz moderner X-by-wire-Technik erfordert strengere Maßnahmen bei der sicherheitsorientierten Systemanalyse. Die Autoren beschreiben eine methodenbasierte Erarbeitung von Überwachungsverfahren und Sicherheitskonzepten, die auf einer System-FMEA basieren und eine Risikoanalyse und übergreifende Fehleranalyse.

Abschließend in diesem Zusammenhang sei [Leve95] empfohlen. Hier werden ausführlich die Grundlagen der Risikountersuchungen in technischen Systemen, die verwendeten Begriffe, Grundlagen der Systemsicherheit sowie Techniken der Risikoanalyse behandelt. Anhand von Unfallbeispielen wird die Rolle von Computern in modernen technischen Systemen deutlich gemacht.

Als besonderes Beispiel eines deduktiven Verfahrens zur Gefahrenanalyse eines technischen Systems wird nachfolgend die ***Fehlerbaumanalyse*** beschrieben. Dies geschieht vor allem auch deshalb, weil mit ihr innerhalb der nachfolgenden Fallstudie besondere Systemanforderungen identifiziert werden können (siehe Abschnitt 5.3.4).

Bei der Fehlerbaumanalyse werden alle Ursachen gesucht, die zu einer Gefährdung führen können. Diese Ereignisse werden durch logische Operatoren (Disjunktion, Konjunktion) verknüpft. Durch die hierarchische Staffelung von Zwischenergebnissen und Folgebedingungen ergibt sich daraus der Fehlerbaum, der die logische, grafische Darstellung des zu untersuchenden Systems ist. Ziel der Fehlerbaumanalyse ist die systematische Identifizierung aller möglichen Ursachen, die zu einem Ausfall führen können. Als eine Besonderheit lassen sich Ausfall- bzw. Eintrittswahrscheinlichkeiten für die einzelnen Ereignisse festlegen, so dass auch eine Gesamtausfallwahrscheinlichkeit für das gesamte System errechnet werden kann.

In der Praxis geht man bei der Fehlerbaumanalyse von der zu vermeidenden Situation aus, diese bildet die Wurzel des Baumes (die Wurzel befindet sich in der grafischen Darstellung jedoch oben). Nun ist zu analysieren, welche Voraussetzungen und Bedingungen eintreten müssten, damit diese gefährliche Situation schließlich eintritt. Sollte es mehrere Vorbedingungen für die Existenz einer Situation geben, so können diese entweder konjunktiv (die aufgeführten Vorbedingungen müssen gleichzeitig erfüllt sein) oder disjunktiv (lediglich eine der Vorbedingungen muss erfüllt sein) verknüpft sein. Auf diese Weise baut sich eine hierarchische Baumstruktur auf. Bezogen auf eine konkrete Anlage oder Maschine sind bei der Analyse die gegebenen konstruktiven und technologischen Zusammenhänge zu betrachten. Die Abbildung 29 stellt ein Beispiel für eine Fehlerbaumanalyse innerhalb der vorgestellten Fallstudie dar.

# 5 Validierung des Prototypen anhand einer Fallstudie

Die folgende Fallstudie dient der Illustration des Sprachumfangs und der Handhabung der Sicherheitsfachsprache und spiegelt die Einsatzmöglichkeiten der SFS in den verschiedenen Bereichen eines automatisierten Systems wider.

Die Fallstudie „Erweiterte Produktionszelle“ basiert auf der bekannten und in der Literatur oft verwendeten Fallstudie „Produktionszelle“ [Lewe95]. Betrachtungsobjekt ist eine reale Produktionsstätte in einem metallverarbeitenden Betrieb. In ihr werden Metallschienen durch eine Presse bearbeitet. Die Anlieferung der Rohteile erfolgt über ein Zuführband und einen Hubdrehtisch, von dort wird die Presse durch einen Roboter beladen (vergleiche Abbildung 25). Nach dem Bearbeitungsvorgang wird die Presse wieder durch den Roboter entladen, die Fertigteile werden durch ein Ablageband abtransportiert. Um einen zyklischen Betrieb zu simulieren, wurde die reale Anlage in der Studie um einen Kran erweitert. Dieser nimmt das Fertigteil am Ende des Ablagebands auf und legt es wieder auf dem Zuführband ab.

## 5.1 Einführung in die Fallstudie

### *5.1.1 Die Gesamtanlage*

Die in der „Erweiterten Produktionszelle“ zusätzlich eingebrachten Komponenten sollen dazu dienen, die ursprüngliche Produktionszelle in eine Produktionsanlage mit einem möglichst realen technischen Hintergrund zu integrieren. Die Gesamtanlage (Abbildung 24) umfasst jetzt mehrere Produktionszellen unterschiedlichen Typs (wobei die ursprüngliche Produktionszelle jetzt als Produktionszelle 1 wiederzufinden ist), ein Transportsystem sowie jeweils ein Lager für Roh- bzw. Fertigteile. Die genaue technische Realisierung dieser zusätzlichen Komponenten sowie der Datenaustausch zwischen den einzelnen Komponenten soll unspezifiziert bleiben.

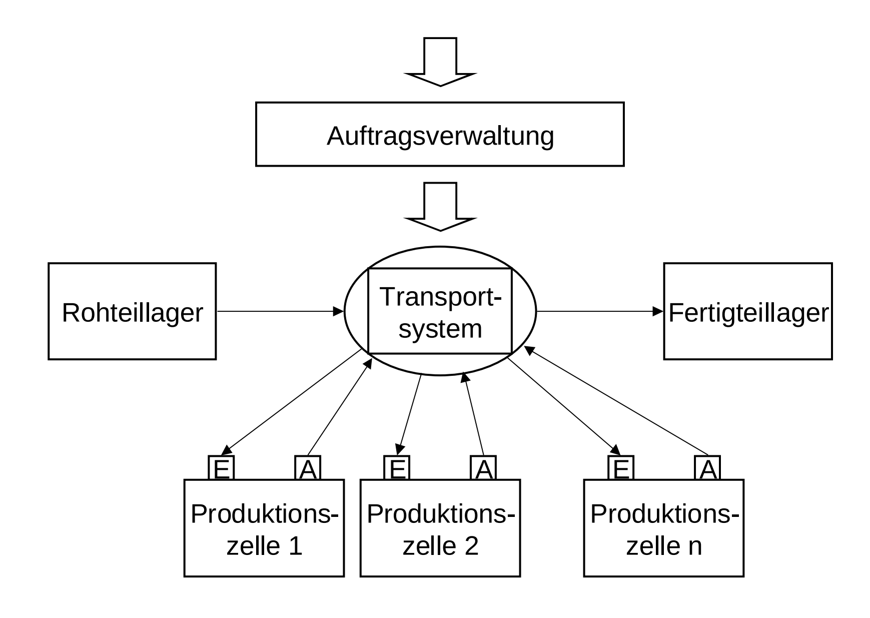



*Abbildung 24 - Schema der „Erweiterten Produktionszelle"*

Mit Hilfe der verschiedenen Produktionszellen ist es möglich, unterschiedliche Bearbeitungsaufträge auszuführen. Durch die zentrale Auftragsverwaltung erfolgen die Koordinierung der einzelnen Bearbeitungsaufträge und die Steuerung des Transportsystems. Dieses holt Rohteile aus dem Rohteillager und bringt diese zu der gewünschten Produktionszelle. Zur Übergabe der Werkstücke besitzt jede Produktionszelle jeweils einen Eingangspuffer (E) und einen Ausgangspuffer (A). Nach Fertigungsabschluss werden die bearbeiteten Teile wieder durch das Transportsystem abgeholt und zum Fertigteillager gebracht.

### *5.1.2 Das Transportsystem*

Das Transportsystem nimmt in der erweiterten Produktionszelle eine zentrale Rolle ein. Es dient dazu, die einzelnen Produktionszellen mit Rohteilen zu versorgen und nach Beendigung des Bearbeitungsprozesses die Fertigteile wieder abzuholen. Die technische Realisierung soll innerhalb dieser Fallstudie jedoch nicht weiter ausgeführt werden, denkbar wäre jedoch ein automatisch und autonom arbeitendes und frei bewegliches Transportsystem. Dazu erhält es Vorgaben aus der übergeordneten zentralen Auftragsverwaltung.

### *5.1.3 Die Materiallager*

Innerhalb der Produktionsanlage gibt es zwei „unendliche“ Materiallager. Im Rohteillager befindet sich jederzeit ein gewisser Vorrat an Rohteilen, in das Fertigteillager kann eine beliebige Menge an Fertigteilen abgelegt werden. Bei der Entnahme von Rohteilen aus dem Rohteillager durch das Transportsystem wird auf eine hier nicht näher bestimmte Art und Weise die gewünschte Anzahl von Rohteilen übergeben. Ebenso erfolgt eine vollständige Übergabe aller Fertigteile in das Fertigteillager.

### *5.1.4 Die Produktionszelle 1*

Die Produktionszelle des ursprünglichen Fallbeispiels wurde erweitert, um den Gegebenheiten der Integration in das Gesamtsystem Rechnung zu tragen und um einen möglichst realen technischen Aufbau zu gewährleisten. Nachdem in der ursprünglichen Produktionszelle der Kran für die Überführung der Werkstücke vom Ablageband zurück zum Zuführband diente, wird diese zyklische Arbeitsweise wieder aufgebrochen. Die Be- und Entladung der Produktionszelle erfolgt über den Eingangs- bzw. den Ausgangspuffer. Außerdem wurde die Produktionszelle 1 um einige Bedien- und Anzeigeelemente erweitert. Es wird wiederum weder Anspruch auf die Vollständigkeit der Elemente noch auf ihre Realisierbarkeit oder Sinnfälligkeit erhoben.

Die Abbildung 25 zeigt den Aufbau der erweiterten Produktionszelle 1.

*Abbildung 25 - Interner Aufbau der erweiterten Produktionszelle 1*

Zu erkennen sind der bereits beschriebene Eingangspuffer und der Ausgangspuffer. Der Kran holt die Rohteile vom Eingangspuffer und legt sie auf dem Zuführband ab. Ebenso nimmt er die Fertigteile vom Ablageband und transportiert sie zum Ausgangspuffer. Alle anderen Bauelemente und Transportvorgänge entsprechen den bereits dargestellten Vorgaben der Produktionszelle.

Die folgenden Bedien- und Anzeigeelemente wurden hinzugefügt:

- Anzeige für die Anzahl der Teile, die dem Eingangspuffer entnommen wurden,
- Anzeige „Anlage bereit“,
- Anzeige „Automatik läuft“,
- Anzeige für die Anzahl der Teile, die im Ausgangspuffer abgelegt wurden.
- Re-Start-Taster (zum Wiederanlauf nach Not-Aus), nichtrastend,
- Not-Aus-Schalter, rastend, nullaktiv,
- Anzeige „Not-Aus“ (in Not-Aus-Schalter integriert).

Mit diesen Darstellungen sind die konstruktiven Gegebenheiten der Produktionszelle 1 und der Gesamtanlage ausreichend beschrieben, auf weitere Bedien- und Anzeigeelemente wird aus Gründen der Übersichtlichkeit verzichtet.

## 5.2 Erläuterungen zur Vorgehensweise bei der Spezifikation

Bevor im Abschnitt 5.3 die formale Spezifikation der „Erweiterten Produktionszelle“ mit Hilfe des SFS-Editors dargestellt wird, sollen an dieser Stelle einige Vorbemerkungen und Erläuterungen erfolgen.

Es wurde bereits dargestellt, dass die formale Darstellung der Programmanforderungen sowohl grundlegender Ausgangspunkt für die Synthese eines neuen Steuerungsprogramms, als auch eine notwendige Voraussetzung für die Verifikation eines bereits existierenden Steuerungsprogramms ist. Prinzipiell unterscheidet sich eine Spezifikation im Rahmen einer Programmsynthese von einer Spezifikation im Zusammenhang mit einer Programmverifikation nur geringfügig (realisierungsunabhängige bzw. realisierungsabhängige Anforderungen - vergleiche Abschnitt 4.5). Die nachfolgende Spezifikation der „Erweiterten Produktionszelle“ soll diesmal unter dem Blickwinkel der Erstellung eines neuen Programms stehen.

Im Verlauf der Spezifikation wird eine Vorgehensweise angewendet, wie sie typisch für komplexe und hierarchisch aufgebaute Systeme ist. Die benutzte Top-Down-Methodik (vgl. [Stet87]) orientiert sich an den gegebenen Strukturen des Gesamtsystems und spiegelt vor allem auch die Aufteilung des Systems in verschiedene Informationsebenen wider. Bei dieser deduktiven Methode wird ausgehend von einer relativ allgemeinen Problemstellung in den oberen Hierarchieebenen eine immer feinere Detaillierung und Präzisierung der Programmanforderungen gezeigt. Ungeachtet dessen wird innerhalb dieser Arbeit kein Anspruch auf die Sinnfälligkeit dieser Methodik erhoben.

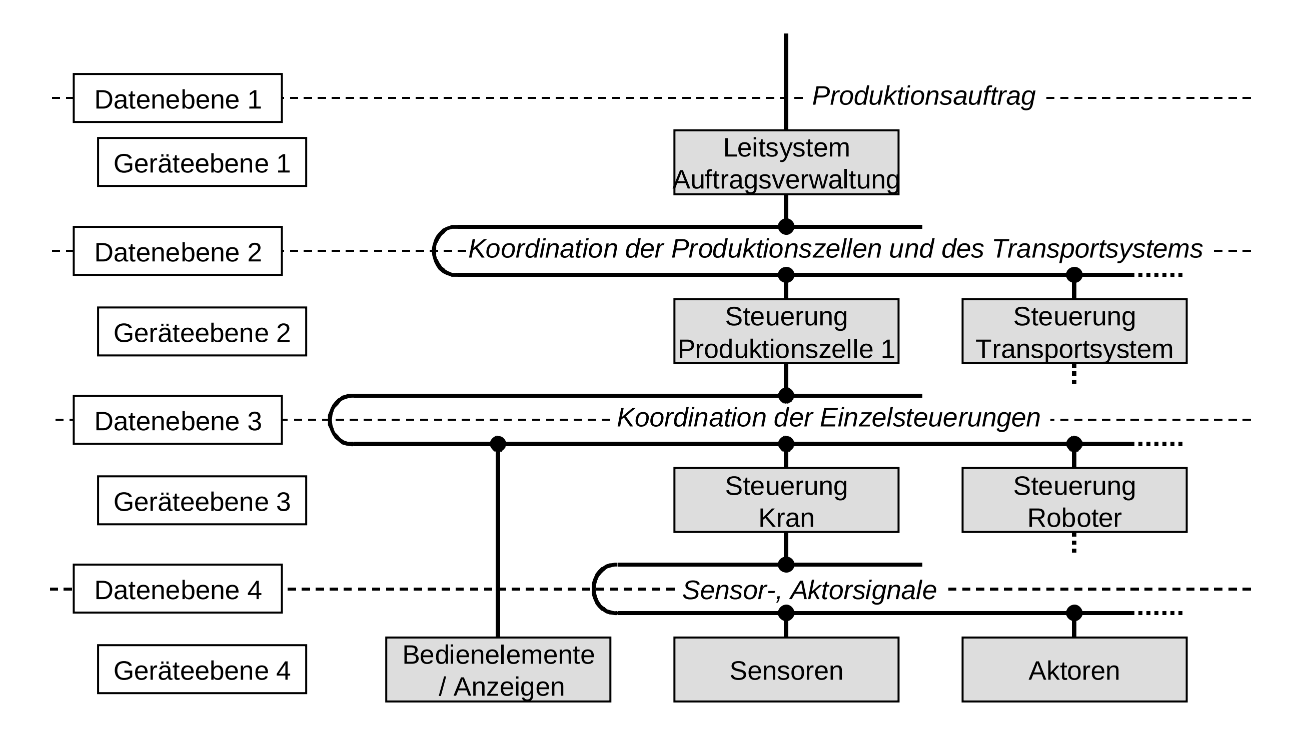



*Abbildung 26 - Gerätetechnischer Aufbau*

Die Abbildung 26 zeigt eine mögliche technische Realisierung der Gesamtanlage. Zu erkennen sind verschiedene Ebenen, wobei sich Datenebenen mit Geräteebenen

abwechseln. In den Geräteebenen befinden sich die einzelnen Steuerungssysteme, wie das Leitsystem, die Gruppen- oder Einzelsteuerungen und schließlich die Sensoren und Aktoren.

Zwischen den Geräteebenen befindet sich jeweils eine Datenebene, durch die spezifische Daten zwischen den Geräteebenen ausgetauscht werden. Technisch betrachtet kann die Realisierung der Datenebenen durch verschiedene Netzwerksysteme, z. B. Ethernet-LAN oder Feldbusse, erfolgen. Die dargestellte Struktur stellt sicherlich nur eine der möglichen Realisierungsvarianten dar. Sie wurde gewählt, um verschiedene typische Automatisierungsstrukturen aufzuzeigen und die weitreichenden Anwendungsmöglichkeiten der Sicherheitsfachsprache zu demonstrieren.

Die Beschreibung der Gesamtanlage ist bis zu diesem Zeitpunkt sehr informell. Außer einer „ungefähren Ahnung“ von der gewünschten Funktionsweise der Gesamtanlage sind kaum konkrete Anforderungen an die einzelnen Elemente und erst recht keine Informationen über auszutauschende Daten vorhanden. Die Zusammenhänge, Abläufe und die informationstechnischen Verbindungen werden sukzessive nach der Top-Down-Methodik aufgebaut. In diesem Zusammenhang werden schrittweise die konkreten Programmanforderungen formuliert und weitere Daten (Bereitschafts-, Fertigmeldungen sowie Koppelsignale) erzeugt.

Dabei wird nach dem folgenden Schema vorgegangen (vgl. Abbildung 19):

1. Zusammenstellung der gegebenen Daten aus der übergeordneten und (falls bereits bekannt) der untergeordneten Datenebene,
2. informelle Spezifikation, Darstellung der technisch/technologische Anforderungen,
3. falls notwendig, erfolgt die Definition neuer Daten aus der informellen Spezifikation,
4. formale Spezifikation der Anforderungen mit der Sicherheitsfachsprache.

Die Punkte 2 bis 4 sind dabei, wenn notwendig, mehrfach zu durchlaufen. Speziell bei einer Programmsynthese ist oftmals ein iteratives Vorgehen notwendig.

Die Anwendung des gezeigten Schemas auf die Fallstudie bringt den folgenden Spezifikationsablauf:

1. Zusammenstellung der gegebenen Daten aus der Datenebene 1
2. Spezifikation der Geräteebene 1 (Auftragsverwaltung)
   - dabei sukzessive Definition von neuen Daten der Datenebene 2 (Transportauftrag, Kommunikation mit Transportsystem und Produktionszellen)
3. Spezifikation der Geräteebene 2 (Mastersteuerung Produktionszelle 1)
   - dabei sukzessive Definition von neuen Daten der Datenebene 3 (Start- und Fertigmeldungen der dezentralen Steuerungen, MMI-Daten)
4. Spezifikation der Geräteebene 2 (Transportsystem)
5. Spezifikation der Geräteebene 3 (Einzelsteuerungen der Zellenkomponenten wie Kran, Transportbänder etc.)
   - dabei sukzessive Definition der verwendeten Aktor- und Sensordaten

## 5.3 Ausführung der Spezifikation

Das Problem wird nun schrittweise unter Beachtung der technischen und technologischen Gegebenheiten zerlegt. Aus Platzgründen wird innerhalb dieser Arbeit auf eine vollständige Spezifikation des Gesamtsystems verzichtet, auf den folgenden Seiten wird lediglich das grundsätzliche Vorgehen dargestellt. Darüber hinaus gehende Informationen können dem Anhang A.6 entnommen werden.

Die Abbildung 27 zeigt, wie - in Analogie zum gerätetechnischen Aufbau - die technologische Aufgabenstellung in den einzelnen Ebenen zerlegt, verfeinert und präzisiert werden kann (aus Platzgründen wurde die Darstellung um 90° gekippt, die oberste Ebene ist jetzt links).

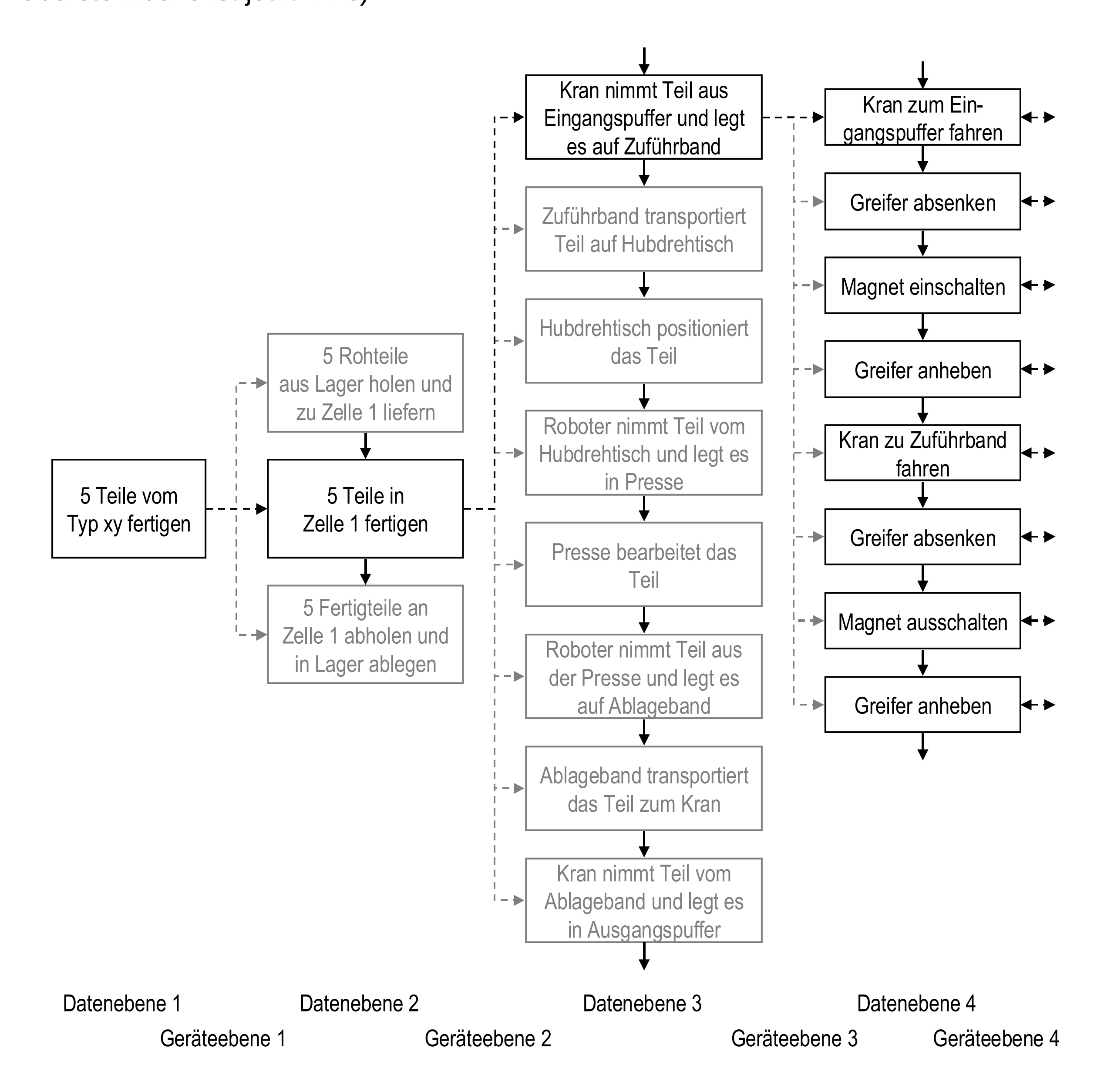



*Abbildung 27 - Abstrakte Problemzerlegung als Funktionsbaum*

Da die zu erstellenden Steuerungsprogramme in den jeweiligen Leitsystemen bzw. Steuerungen abgearbeitet werden, erfolgt nun die Darstellung der Spezifikation für ausgewählte Steuerungskomponenten in den dargestellten Geräteebenen unter Berücksichtigung der angrenzenden Datenebenen.

### *5.3.1 Geräteebene 1 - Die Auftragsverwaltung*

Die Produktionsanlage der Fallstudie „Erweiterte Produktionszelle“ besteht aus verschiedenen Objekten, deren koordinierter Einsatz eine möglichst effektive und korrekte Produktion verschiedener Fertigteile ermöglichen soll.

Die eingehenden Produktionsaufträge müssen innerhalb der Auftragsverwaltung gesammelt und für die untergeordnete Ebene in geeigneter Weise aufbereitet werden. Jede Produktionszelle kann nur eine bestimmte Bearbeitungsart (z. B. Bohren, Pressen, Fräsen usw.) ausführen, die Zuordnung von Bearbeitungsart und Produktionszelle ist eineindeutig. Zur Steigerung der Effektivität der Gesamtanlage soll es möglich sein, mehrere Produktionsaufträge gleichzeitig zu bearbeiten, d. h. Produktionsaufträge werden entsprechend der Auslastung der Produktionszellen unmittelbar gestartet.

*Gegebene Daten:*

Die Auftragsverwaltung erhält von einer übergeordneten, nicht näher spezifizierten Ebene neue Produktionsaufträge. Weitere konkrete Daten sind in diesem Zusammenhang nicht bekannt.

*Informelle Spezifikation:*

Es müssen Produktionsaufträge der folgenden Art spezifiziert werden: „Es sollen n Teile vom Typ xy gefertigt werden.“

*Definition neuer Daten mit dem SFS-Editor:*

Es werden die folgenden Variablen definiert:

- `order_number` (Auftragsnummer)
- `order_quantity` (Anzahl der zu fertigenden Teile)
- `order_work` (Art der zu fertigenden Teile)
- `order_status` (Bearbeitungsstatus des Auftrags)

Der detaillierte Aufbau eines Produktionsauftrages sowie die Verbalphrasen der Nomen und Werte können der Tabelle 9 (Anhang, Seite 140) entnommen werden.

### Bearbeitung der Produktionsaufträge

*Aufgabe:*

Eingehende Produktionsaufträge müssen schnellstmöglich und vollständig bearbeitet werden. Innerhalb des Produktionsauftrages nimmt die soeben definierte Variable `order_status` eine herausragende Rolle ein. Der aktuelle Wert muss durch die Auftragsverwaltung durch geeignete Kommunikation mit den anderen Elementen der Produktionsanlage bestimmt werden.

*Informelle Spezifikation:*

Produktionsaufträge, die den Status „Neu“, „Rohteile liefern“ usw. haben, müssen irgendwann den Status „Fertig“ haben.

*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 1<<< (einfache Forderung - unbegrenzt) */*
*Wenn irgendwann "der aktuelle Stand der Auftragsbearbeitung" "Neu" war , dann muss "der aktuelle Stand der Auftragsbearbeitung" irgendwann "auf Fertig gesetzt" werden .*

```
AG ( (rdy_in & order_status=1) -> AF (rdy_plc &
order_status=5) )
```

Analog zum dargestellten Satz können weitere Anforderungen definiert werden (`order_status` = 2, 3 oder 4)

### Aufbereitung der Produktionsaufträge und Kommunikation mit der Geräteebene 2

Die Auftragsverwaltung kommuniziert in geeigneter Weise mit dem Transportsystem und den Produktionszellen, um die Ausführung der Produktionsaufträge zu gewährleisten.

Neben der Verwaltung der Produktionsaufträge besteht eine wesentliche Aufgabe der Auftragsverwaltung in einer gezielten Kommunikation mit dem Transportsystem und den Produktionszellen, um die einzelnen Produktionsaufträge auszuführen. Durch die Analyse eines Produktionsauftrages ergeben sich die folgenden Kommunikationsbeziehungen:

- Kommunikation mit dem Transportsystem, um Rohteile aus dem Lager zu einer Produktionszelle zu befördern,
- Kommunikation mit der Produktionszelle, um die erforderlichen Bearbeitungsvorgänge auszulösen,
- nochmalige Kommunikation mit dem Transportsystem, um die Fertigteile von der Produktionszelle zum Lager zu transportieren.

*Informelle Spezifikation:*

Es wurde dargestellt, dass die Auftragsverwaltung in geeigneter Weise mit dem Transportsystem kommunizieren und dieses zum Bestücken der Produktionszellen mit Rohteilen bzw. zum Entladen der Produktionszellen veranlassen soll.

*Definition neuer Daten mit dem SFS-Editor:*

Es wird ein Datensatz „Transportauftrag“ definiert. Durch diesen wird das Transportsystem veranlasst, entweder Rohteile oder Fertigteile zu transportieren bzw. in Ruhe zu verharren. Details des Transportauftrags können der Tabelle 10 (Anhang, Seite 142) entnommen werden.

*Aufgabe:*

Die Auslösung eines Transportauftrages muss mit der jeweiligen Produktionszelle und dem Transportsystem koordiniert werden. Das Transportsystem kann einen neuen Transportauftrag nur ausführen, wenn es frei, d. h. unbeschäftigt ist. Die Produktionszellen können nur beladen werden, wenn sie dazu bereit sind, und sie können erst entladen werden, wenn der aktuelle Produktionsauftrag beendet wurde.

*Definition neuer Daten mit dem SFS-Editor:*

Für jede Produktionszelle sowie für das Transportsystem werden Bereitschafts- bzw. Fertigmeldungen definiert, die den aktuellen Zustand dieser Elemente anzeigen.

- `cell_1_ready` (Produktionszelle 1 ist für neuen Auftrag bereit)
- `cell_1_finished` (Produktionszelle 1 hat den Auftrag bearbeitet)
- ... (analog für die weiteren Produktionszellen)
- `tr_sys_ready` (Transportsystem ist für neuen Auftrag bereit)
- `tr_sys_finished` (Transportsystem hat Auftrag ausgeführt)

Details zu diesen Meldungen können der Tabelle 11 (Anhang, Seite 143), Tabelle 12 (Anhang, Seite 143) und Tabelle 13 (Anhang, Seite 144) entnommen werden.

Die Unterteilung in Bereitschafts- und Fertigmeldungen wurde vorgenommen, weil sich somit durch logische Verknüpfung dieser Signale weitere Informationen übertragen lassen (z. B. *„mit altem Auftrag fertig“* UND nicht *„bereit für neuen Auftrag“* bedeutet Defekt bzw. Störung)

*Aufgabe:*

Die Transportaufträge können nur unter Beachtung von Bereitschafts- und Fertigmeldungen der Produktionszellen bzw. des Transportsystems ausgegeben werden.

*Informelle Spezifikation:*

Eine Beladung mit Rohteilen erfolgt, wenn es einen Produktionsauftrag für die jeweilige Produktionszelle gibt, wenn die Bereitschaftsmeldung dieser Produktionszelle gesetzt ist und wenn die Bereitschaftsmeldung des Transportsystems gesetzt ist.

Eine Entladung der Produktionszelle erfolgt, wenn die Fertigmeldung einer Produktionszelle gesetzt ist und wenn die Bereitschaftsmeldung des Transportsystems gesetzt ist.

*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 2<<< (einfache Forderung - direkt) */*

*Wenn "der aktuelle Stand der Auftragsbearbeitung" "Neu" ist und "die Bearbeitungsart" ist "Pressen" und "die Bereitschaftsmeldung der Produktionszelle 1" ist "gesetzt" und "die Bereitschaftsmeldung des Transportsystems" ist "gesetzt" , dann muss "der aktuelle Stand der Auftragsbearbeitung" unmittelbar "auf Rohteile holen gesetzt" werden und "die Transportart" muss unmittelbar "auf 1 (Rohteile holen) gesetzt" werden und "die anzufahrende Produktionszelle" muss unmittelbar "1" werden und "die Bereitschaftsmeldung der Produktionszelle 1" muss unmittelbar "zurückgesetzt" werden und "die Bereitschaftsmeldung des Transportsystems" muss unmittelbar "zurückgesetzt" werden .*

```
AG ( (rdy_in & order_status=1 & order_work=1 &
cell_1_ready & tr_sys_ready) -> A[!rdy_plc U (rdy_plc &
order_status=2 & tr_kind=1 & tr_place=1 & !cell_1_ready &
!tr_sys_ready) ] )
```

*/* >>>Satz 3<<< (einfache Forderung - direkt) */*

Wenn "der aktuelle Stand der Auftragsbearbeitung" "Bearbeiten" ist und "die Fertigmeldung der Produktionszelle 1" ist "gesetzt" und "die Bereitschaftsmeldung des Transportsystems" "gesetzt" ist , dann muss "der aktuelle Stand der Auftragsbearbeitung" unmittelbar "auf Fertigteile wegbringen gesetzt" werden und "die Transportart" muss unmittelbar "auf 2 (Fertigteile wegbringen) gesetzt" werden und "die anzufahrende Produktionszelle" muss unmittelbar "1" werden und "die Fertigmeldung der Produktionszelle 1" muss unmittelbar "zurückgesetzt" werden und "die Bereitschaftsmeldung des Transportsystems" muss unmittelbar "zurückgesetzt" werden .

```
AG ( (rdy_in & order_status=3 & cell_1_finished &
tr_sys_ready) -> A[!rdy_plc U (rdy_plc & order_status=4 &
tr_kind=2 & tr_place=1 & !cell_1_finished & !tr_sys_ready)
] )
```

Analog zu den dargestellten Sätzen können weitere Anforderungen definiert werden.

*Aufgabe:*

In Anlehnung an die soeben formulierten Anforderungen dürfen Transportaufträge jedoch nicht ausgelöst werden, wenn die betreffende Produktionszelle nicht bereit ist.

*Informelle Spezifikation:*

Eine Beladung mit Rohteilen darf nicht erfolgen, wenn die Bereitschaftsmeldung einer Produktionszelle nicht gesetzt ist. Eine Entladung der Produktionszelle darf nicht erfolgen, wenn die Fertigmeldung einer Produktionszelle nicht gesetzt ist.

*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 7<<< (einfaches Verbot - Zustand) */*

*Wenn "die Bereitschaftsmeldung der Produktionszelle 1" "nicht gesetzt" ist , dann darf "die Transportart" nicht gleichzeitig "auf 1 Rohteile holen gesetzt" sein und "die anzufahrende Produktionszelle" darf nicht gleichzeitig "1" sein .*

```
AG !( rdy_plc & !cell_1_ready & (tr_kind=1 | tr_place=1) )
```

*/* >>>Satz 8<<< (einfaches Verbot - Zustand) */*

*Wenn "die Fertigmeldung der Produktionszelle 1" "nicht gesetzt" ist , dann darf "die Transportart" nicht gleichzeitig "auf 2 Fertigteile wegbringen gesetzt" sein und "die anzufahrende Produktionszelle" darf nicht gleichzeitig "1" sein .*

```
AG !( rdy_plc & !cell_1_finished & (tr_kind=2 |
tr_place=1) )
```

### 5.3.2 Geräteebene 2 – Steuerung der Produktionszelle 1

Innerhalb dieser Arbeit wird nur die Produktionszelle 1 betrachtet. Es handelt sich hierbei um diejenige, die Bestandteil der ursprünglichen Fallstudie war. Es wird darauf verwiesen, dass die anderen Produktionszellen der Anlage ähnlich aufgebaut sind und die Spezifikation mit denselben Methoden und Verfahren erfolgen kann.

Das Steuerungssystem der Produktionszelle wurde als hierarchisches Steuerungssystem aufgebaut. Jede Komponente innerhalb der Produktionszelle (Kran, Roboter, Presse usw.) ist mit einer separaten Steuerung ausgestattet. Die zentrale „Master“-Steuerung, die in dieser Geräteebene 2 beschrieben wird, koordiniert die dezentralen Steuerungen, so dass der geforderte Bearbeitungsablauf realisiert wird (vergleiche Abbildung 28). Sobald erkannt wird, dass im Eingangspuffer ein Rohteil liegt, wird dieses durch den Kran aufgenommen und auf das Zuführband gelegt. Das Zuführband transportiert das Teil auf den Hubdrehtisch. Dieser vollführt eine Drehung und hebt das Teil gleichzeitig etwas an, so dass sich das Teil anschließend in einer Position befindet, in der es vom Roboterarm 1 aufgenommen werden kann. Der Roboterarm 1 legt das Teil in der Presse ab. Diese presst anschließend das Teil. Das Fertigteil wird durch den Roboterarm 2 aus der Presse genommen und auf dem Ablage-

band abgelegt. Dieses befördert das Teil zum Kran, der anschließend das Teil wieder vom Ablageband nimmt und in den Ausgangspuffer legt.

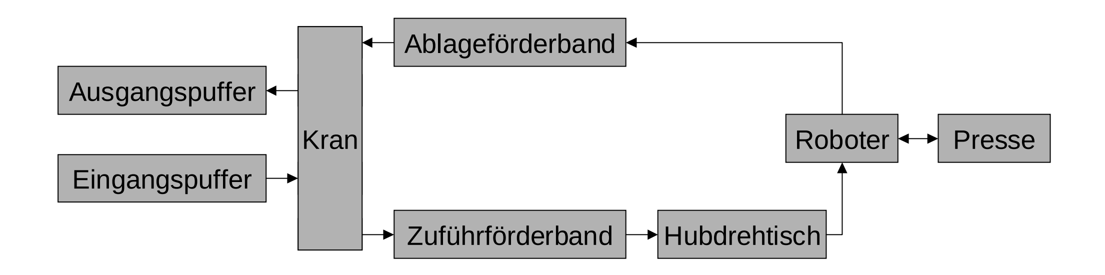



*Abbildung 28 - Materialfluss in der Produktionszelle*

Diese Beschreibung stellt einen vollständigen und sicheren, jedoch nicht unbedingt effizienten Bearbeitungsvorgang dar. Im Anhang sind mehrere Möglichkeiten dargestellt, um den Durchsatz von Werkstücken zu erhöhen. Die hierzu nutzbaren Ansätze zielen auf eine Verfeinerung der einzelnen Transportvorgänge sowie die Koordinierung einzelner Transport- und Übergabevorgänge ab.

*Aufgabe:*

Die zentrale Steuerung der Produktionszelle 1 muss zwei Kommunikationsaufgaben erfüllen:

- Kommunikation mit der übergeordneten Auftragsverwaltung: Bereitstellung der Bereitschafts- bzw. Fertigmeldung,
- Kommunikation mit den untergeordneten Steuerungen der Einzelkomponenten.

Darüber hinaus erfolgt die Ankopplung der Bedien- und Anzeigeelemente der Produktionszelle 1 direkt an die zentrale Steuerung.

*Gegebene Daten:*

- bereits definierte Bereitschafts- und Fertigmeldung der Produktionszelle (siehe Tabelle 11 und Tabelle 12, Anhang, Seite 143),
- Sensor- und Aktorsignale der Bedien- bzw. Anzeigeelemente (siehe Tabelle 14, Anhang, Seite 146 und Tabelle 15, Anhang, Seite 147).

Die nachfolgenden Ausführungen beziehen sich ausschließlich auf die Kommunikation der zentralen Steuerung mit der Steuerung des Krans, die Kommunikation mit den Steuerungen der anderen Komponenten der Geräteebene 3 erfolgt analog und kann dem Anhang entnommen werden.

**Kommunikation mit der Steuerung des Krans**

Der Kran hat innerhalb der Produktionszelle zwei Aufgaben:

- Transport von Rohteilen vom Eingangspuffer zum Zuführband,
- Transport von Fertigteilen vom Ablageband zum Ausgangspuffer.

Diese Aufgaben können nur alternativ bearbeitet werden, da der Kran immer nur ein Teil transportieren kann. Nachfolgend wird nur der Transport der Rohteile betrachtet.

*Gegebene Daten:*

- bereits definiertes Not-Aus-Signal für die Steuerung des Krans,
- Sensorwert für die Belegung des Eingangspuffers.

*Informelle Spezifikation:*

Die Steuerung der Produktionszelle muss an die Steuerung des Krans das Signal geben, wenn dieser ein Rohteil aus dem Eingangspuffer holen soll. Dazu müssen der Zellensteuerung die folgenden Informationen bekannt sein:

- im Eingangspuffer liegt ein Rohteil,
- das Rohteil kann auf dem Zuführband abgelegt werden.

Während die erste Information ein bereits bekanntes Sensorsignal ist, ist die letztere Information noch unbekannt.

*Definition neuer Daten mit dem SFS-Editor:*

Das Zuführband muss eine Meldung generieren, dass es zur Übernahme eines neuen Teils bereit ist. Weiterhin wird ein Startsignal definiert, das den Kran zum Transport eines Rohteils vom Eingangspuffer zum Zuführband veranlasst.

- `Fbelt_rdy_take` (Zuführband ist zur Aufnahme eines Teils bereit)
- `Crane_run_source` (Kran soll Rohteil holen)

Die Details können wieder der Tabelle 24 (Anhang, Seite 176) sowie der Tabelle 19 (Anhang, Seite 159) entnommen werden.

*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 23<<< (einfache Forderung - direkt) */*

*Wenn "die Produktionszelle 1" "im Automatikbetrieb" ist und "der Eingangspuffer" ist "belegt" und "die Bereitschaftsmeldung 'Zuführband ist zur Übernahme eines Teils bereit'" ist "gesetzt" , dann muss "die Bereitschaftsmeldung 'Zuführband ist zur Übernahme eines Teils bereit'" unmittelbar "zurückgesetzt" werden und "das Startsignal 'Kran holt Rohteil von Eingangspuffer'" muss unmittelbar "gesetzt" werden .*

```
AG ( (rdy_in & cell_1_state=1 & Piece_at_source &
Fbelt_rdy_take) -> A[!rdy_plc U (rdy_plc & !Fbelt_rdy_take
& Crane_run_source) ] )
```

*weitere informelle Spezifikationen:*

Es ist nicht sinnvoll, den Kran zu starten, wenn der Eingangspuffer leer ist, d. h. keine Rohteile vorhanden sind.

Außerdem darf der Kran dann auch nur gestartet werden, wenn er ein Teil auf dem Zuführband ablegen kann. Dieses muss also dazu bereit sein.

*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 24<<< (erweiterte Möglichkeit - Zustand) */*

*"das Startsignal 'Kran holt Rohteil vom Eingangspuffer'" darf nur "gesetzt" sein , wenn gleichzeitig "der Eingangspuffer" "belegt" ist .*

```
AG !( rdy_plc & !(Piece_at_source) & (Crane_run_source) )
```

*/* >>>Satz 25<<< (erweiterte Möglichkeit - Zustand) */*

*"das Startsignal 'Kran holt Rohteil vom Eingangspuffer'" darf nur "gesetzt" sein , wenn gleichzeitig "die Bereitschaftsmeldung 'Zuführband ist zur Übernahme eines Teils bereit'" "gesetzt" ist .*

```
AG !( rdy_plc & !(Fbelt_ready_take) & (Crane_run_source) )
```

### 5.3.3 Geräteebene 3 - Der Kran

Bei Beschreibung der Spezifikation für die Komponenten der Geräteebene 3 wird ausschließlich der Kran betrachtet, die Spezifikation der anderen Komponenten erfolgt analog und kann dem Anhang entnommen werden.

*Gegebene Daten:*

Die Steuerung des Krans erhält von der übergeordneten Zellensteuerung Startsignale zur Ausführung bestimmter Aktionen. Durch verschiedene Sensoren und Aktoren ist die Kransteuerung mit den Maschinenelementen verbunden.

- aus der Datenebene 3 sind die Kommunikationssignale bekannt, siehe Tabelle 24 (Anhang, Seite 176) und Tabelle 19 (Anhang, Seite 159)
- in der Datenebene 4 sind die Sensoren bekannt: siehe Tabelle 25 (Anhang, Seite 181)
- in der Datenebene 4 sind die Aktoren bekannt: siehe Tabelle 26 (Anhang, Seite 181)

*Aufgabe:*

Der Steuerung des Krans verbleibt solange in Ruhe, bis durch die Zellensteuerung entweder das Startsignal zum Transport eines Rohteils oder eines Fertigteils gesetzt wird. Danach wird die geforderte Transportroutine bearbeitet.

In der Abbildung 27 wurde beispielhaft für den Transport eines Rohteils vom Eingangspuffer zum Zuführband die Verfeinerung dieses Transportvorgangs dargestellt.

*Informelle Spezifikation:*

Damit der Kran den Eingangspuffer korrekt anfahren kann, muss er zunächst seine relative Position zu ihm bestimmen. Erst dann kann entschieden werden, in welche Richtung der Kran fahren soll, d. h. welches Aktorsignal zu aktiveren ist. Prinzipiell kann davon ausgegangen werden, dass sich der Kran nach einem zuvor erfolgten Transportvorgang entweder am Zuführband oder am Ausgangspuffer befindet. Diese beiden Positionen, und auch wenn sich der Kran bereits am Eingangspuffer oder am Ablageband befindet, können durch die vorhandenen Sensoren erkannt werden. Sollte sich der Kran an einer unbekannten Position befinden (wenn also keiner der vier Sensoren aktiv ist), so muss zunächst eine Fahrtrichtung festgelegt und durch nachfolgende erstmalige Erkennung eines Sensors die Position bestimmt werden. Unter Umständen ist dann die Fahrtrichtung wieder umzukehren. Hat der Kran jedoch die Position des Eingangspuffers erreicht, so wird der Magnetgreifer auf die Höhe des Eingangspuffers abgesenkt. Dann wird der Magnetgreifer aktiviert. Danach beginnt das Anheben des Greifers. Sobald dieser wieder oben ist, fährt der Kran zum Zuführband. Dort angekommen, wird der Greifer wieder abgesenkt. Auf der Höhe des Zuführbands wird der Greifer angehalten und der Magnet wird abgeschalten. Gleichzeitig kann die Meldung „Kran hat Teil auf Zuführband gelegt“ gesetzt werden. Danach wird der Greifer wieder angehoben. Sobald der Greifer wieder in seiner oberen Position ist, wird er gestoppt und gelangt wieder in den Ruhezustand.

Da die gezeigten Teilvorgänge unbedingt in der festgelegten Reihenfolge abzuarbeiten sind, bietet sich die Entwicklung einer Ablaufstruktur an.

*Definition neuer Daten mit dem SFS-Editor:*

Es wird eine Zustandsvariable definiert, die den aktuellen Zustand darstellt, in dem sich der Kran bzw. die Ablaufsteuerung des Krans befindet:

- `Crane_state` - Zustandsvariable des Krans, siehe Tabelle 27 (Anhang Seite 183)

*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 64<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Krans" "0" ist und "das Startsignal 'Kran holt Rohteil vom Eingangspuffer'" ist "gesetzt" und "das Startsignal 'Kran holt Fertigteil von Ablageband'" ist "nicht gesetzt" , dann muss "das Startsignal 'Kran holt Rohteil vom Eingangspuffer'" unmittelbar "zurückgesetzt" werden und "die Zustandsvariable des Krans" muss unmittelbar "1" werden .*

```
AG ( (rdy_in & Crane_state=0 & Crane_run_source &
!Crane_run_dbelt) -> A[!rdy_plc U (rdy_plc &
!Crane_run_source & Crane_state=1) ] )
```

*/* >>>Satz 65<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Krans" "0" ist und "das Startsignal 'Kran holt Fertigteil von Ablageband'" ist "gesetzt" , dann muss "das Startsignal 'Kran holt Fertigteil von Ablageband'" unmittelbar "zurückgesetzt" werden und "die Zustandsvariable des Krans" muss unmittelbar "2" werden und "die Bewegung des Krans nach rechts (Ablageband)" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Crane_state=0 & Crane_run_dbelt) ->
A[!rdy_plc U (rdy_plc & !Crane_run_dbelt & Crane_state=2 &
Crane_to_dbelt) ] )
```

*/* >>>Satz 66<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Krans" "1" ist und "der Kran" ist "am Eingangspuffer" , dann muss "die Zustandsvariable des Krans" unmittelbar "11" werden und "das Absenken des Krangreifers" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Crane_state=1 & Crane_at_source) ->
A[!rdy_plc U (rdy_plc & Crane_state=11 & Crane_lower) ] )
```

…

…

*Weitere informelle Spezifikationen:*

Neben den Funktionsanforderungen müssen durch die Kransteuerung weitere Sicherheitsanforderungen realisiert werden. Hierbei können die folgenden sicherheitskritischen Punkte identifiziert werden:

- Begrenzung der Kranbewegung an den Endpunkten der Kranbahn
- Begrenzung der Greiferbewegung an den Endpunkten unter Berücksichtigung der Position des Krans (Höhe der Förderbänder sowie des Eingangs- bzw. Ausgangspuffers),
- Der Greifer des Krans darf nur an bestimmten (sicheren) Orten deaktiviert werden.

*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 86<<< (einfaches Verbot - Zustand) */*

*"die Bewegung des Krans nach links (zum Zuführband)" darf nicht gleichzeitig "gestartet" sein , wenn "der Kran" "am Zuführband" ist .*

```
AG !( rdy_plc & Crane_at_fbelt & (Crane_to_fbelt) )
```

/* >>>Satz 87<<< (einfaches Verbot - Zustand) */

"die Bewegung des Krans nach rechts (Ablageband)" darf nicht gleichzeitig "gestartet" sein , wenn "der Kran" "am Ablageband" ist .

```
AG !( rdy_plc & Crane_at_dbelt & (Crane_to_dbelt) )
```

/* >>>Satz 88<<< (einfaches Verbot - Zustand) */

"das Anheben des Krangreifers" darf nicht gleichzeitig "gestartet" sein , wenn "der Greifer des Krans" "oben" ist .

```
AG !( rdy_plc & Crane_height=0 & (Crane_lift) )
```

/* >>>Satz 89<<< (einfaches Verbot - Zustand) */

"das Absenken des Krangreifers" darf nicht gleichzeitig "gestartet" sein , wenn "der Greifer des Krans" "unten" ist .

```
AG !( rdy_plc & Crane_height=1.0 & (Crane_lower) )
```

/* >>>Satz 90<<< (einfaches Verbot - Zustand) */

"das Absenken des Krangreifers" darf nicht gleichzeitig "gestartet" sein , wenn "der Kran" "am Ablageband" ist und "der Greifer des Krans" "auf dem Ablageband" ist .

```
AG !( rdy_plc & Crane_at_dbelt & Crane_height=0.66 &
(Crane_lower) )
```

/* >>>Satz 91<<< (einfaches Verbot - Zustand) */

"das Absenken des Krangreifers" darf nicht gleichzeitig "gestartet" sein , wenn "der Kran" "am Eingangspuffer" ist und "der Greifer des Krans" "auf dem Eingangspuffer" ist .

```
AG !( rdy_plc & Crane_at_source & Crane_height=0.85 &
(Crane_lower) )
```

/* >>>Satz 92<<< (einfaches Verbot - Zustand) */

"das Absenken des Krangreifers" darf nicht gleichzeitig "gestartet" sein , wenn "der Kran" "am Ausgangspuffer" ist und "der Greifer des Krans" "auf dem Ausgangspuffer" ist .

```
AG !( rdy_plc & Crane_at_drain & Crane_height=0.35 &
(Crane_lower) )
```

*/* >>>Satz 93<<< (einfaches Verbot - Zustand) */*

*"das Absenken des Krangreifers" darf nicht gleichzeitig "gestartet" sein , wenn "der Kran" "am Zuführband" ist und "der Greifer des Krans" "auf dem Zuführband" ist .*

```
AG !( rdy_plc & Crane_at_fbelt & Crane_height=0.94 &
(Crane_lower) )
```

*/* >>>Satz 94<<< (einfaches Verbot - selbstbegrenzt) */*

*Solange "der Magnet des Krangreifers" "aktiv" ist und "der Kran" ist "nicht am Ablageband" und "der Kran" ist "nicht am Zuführband" und "der Kran" ist "nicht am Eingangspuffer" und "der Kran" ist "nicht am Ausgangspuffer" , darf "der Magnet des Krangreifers" niemals "deaktiviert" werden .*

```
AG !( rdy_plc & Crane_mag_on & !Crane_at_dbelt &
!Crane_at_fbelt & !Crane_at_source & !Crane_at_drain &
(!Crane_mag_on) )
```

Die Fallstudie würde an dieser Stelle mit der Spezifikation der anderen Teilgeräte (Zuführband, Hubdrehtisch usw.) fortfahren. Diese Spezifikationen unterscheiden sich jedoch nicht grundsätzlich, sondern nur in speziellen Details von der Spezifikation des Krans. Aus diesem Grund wird an dieser Stelle auf diese Darstellung verzichtet, die Details können dem Anhang entnommen werden.

### *5.3.4 Definition weiterer Eigenschaften*

Neben der Betrachtung der Einzelsteuerungen hinsichtlich der Einhaltung von Sicherheitsanforderungen müssen auch übergeordnete Zusammenhänge zwischen den einzelnen Komponenten betrachtet werden. Im Abschnitt 5.3.2 wurden bereits einige Anforderungen im Zusammenhang mit der Sicherung der korrekten Übergabe der Werkstücke zwischen den einzelnen Zellenelementen formuliert.

Auch in der ursprünglichen Fallstudie wurden bereits einige Sicherheitsaspekte benannt, die vom Steuerungsprogramm unbedingt einzuhalten sind. Diese wurden vier Prinzipien zusammengefasst:

1. Begrenzung der Maschinenbeweglichkeit, z. B. Vermeidung der Beschädigung der Presse, falls sie sich zu weit öffnen würde,
2. Vermeidung von Maschinenkollisionen, z. B. der Roboterarme mit der Presse, falls deren Bewegung nicht exakt koordiniert ist,
3. die Werkstücke sollen innerhalb so genannter sicherer Bereiche abgelegt werden, z. B. darf der Magnet des Krangreifers nur an bestimmten Stellen deaktiviert werden.
4. ausreichende Trennung der Werkstücke, d. h. durch die Funktion der Lichtschranken auf den Förderbändern ist ein bestimmter Mindestabstand der Werkstücke für die sichere Erkennung notwendig.

Auch diese informellen Anforderungen können mit der Sicherheitsfachsprache formuliert werden.

Ein weiterer Problembereich zur Sicherung der Gesamtanlage ist die Vermeidung von Zusammenstößen bzw. Kollisionen zwischen verschiedenen beweglichen Elementen der Produktionszelle. Beispiele hierfür sind:

- Kollision des Krangreifers mit dem Zuführband, dem Ablageband, dem Eingangspuffer oder dem Ausgangspuffer,
- Kollision eines Roboterarms mit der Presse,
- Kollision des Pressenwerkzeugs mit einem Roboterarm.

Nachfolgend soll beispielhaft mit Hilfe der in Abschnitt 4.6 dargestellten Fehlerbaumanalyse gezeigt werden, welche Voraussetzungen und Zusammenhänge zu einer Kollision der Roboterarme mit der Presse führen können. Die erzielten Analyseergebnisse fließen anschließend in zusätzliche Programmanforderungen ein.

Als zu verhinderndes Ereignis befindet sich die Kollision eines Roboterarms mit der Presse in der Wurzel des Baums (Abbildung 29).

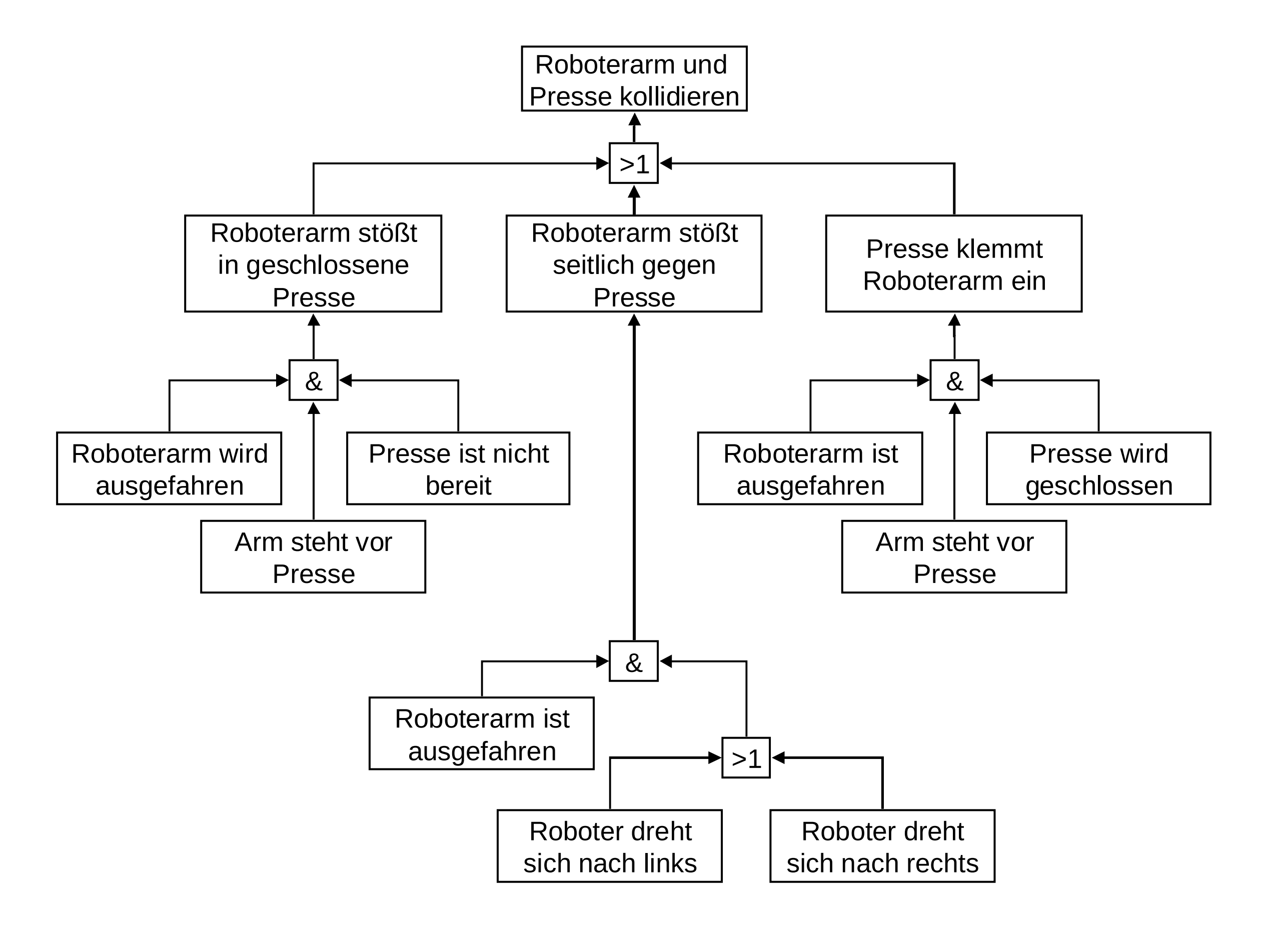



*Abbildung 29 - Fehlerbaumanalyse für die Kollision des Roboters mit der Presse*

Das genannte unerwünschte Ereignis könnte durch die darunter aufgezeigten Ursachen eintreten. Diese einzelnen Ursachen werden solange aufgeschlüsselt, bis sich

die elementaren Ursachen ergeben. Aus diesen erfolgt die Zusammenstellung der informellen Spezifikation der Sicherheitseigenschaften.

*Informelle Spezifikation:*

Ausgehend von Abbildung 29 lassen sich die folgenden Situationen identifizieren, die unbedingt zu vermeiden sind:

- der Roboterarm 1 darf nicht ausfahren, wenn der Roboter mit dem Arm 1 vor der Presse und die Presse nicht in der erforderlichen Position ist,
- der Roboterarm 2 darf nicht ausfahren, wenn der Roboter mit dem Arm 2 vor der Presse und die Presse nicht in der erforderlichen Position ist,
- der Roboter darf sich nicht nach links drehen, wenn der Roboterarm 1 ausgefahren ist,
- der Roboter darf sich nicht nach rechts drehen, wenn der Roboterarm 1 ausgefahren ist,
- der Roboter darf sich nicht nach links drehen, wenn der Roboterarm 2 ausgefahren ist,
- der Roboter darf sich nicht nach rechts drehen, wenn der Roboterarm 2 ausgefahren ist,
- die Presse darf sich nicht schließen, wenn der Roboter mit dem Roboterarm 1 vor der Presse steht und der Roboterarm 1 ausgefahren ist,
- die Presse darf sich nicht schließen, wenn der Roboter mit dem Roboterarm 2 vor der Presse steht und der Roboterarm 2 ausgefahren ist.

*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 155<<< (einfaches Verbot - Zustand) */*

*Wenn "der Roboter" "mit dem Arm 1 vor der Presse" ist und "die Bereitschaftsmeldung 'Presse ist für Beladung bereit'" ist "nicht gesetzt" , dann darf "das Ausfahren des Roboterarms 1" nicht gleichzeitig "gestartet" sein .*

```
AG !( rdy_plc & Robot_angle=-90 & !Press_ready_take &
(Arm1_forward) )
```

*/* >>>Satz 156<<< (einfaches Verbot - Zustand) */*

*Wenn "der Roboter" "mit dem Arm 2 vor der Presse" ist und "die Bereitschaftsmeldung 'Presse ist zur Übergabe eines Teils bereit'" ist "nicht gesetzt" , dann darf "das Ausfahren des Roboterarms 2" nicht gleichzeitig "gestartet" sein .*

```
AG !( rdy_plc & Robot_angle=0 & !Press_ready_give &
(Arm2_forward) )
```

*/* >>>Satz 159<<< (einfaches Verbot - selbstbegrenzt) */*

*"die Drehung des Roboters nach links" darf niemals "gestartet" werden , solange "der*

*Roboterarm 1" "vorn" ist .*

```
AG !( rdy_plc & Arm1_ext=1.0 & (Robot_left) )
```

*/* >>>Satz 160<<< (einfaches Verbot - selbstbegrenzt) */*

*"die Drehung des Roboters nach rechts" darf niemals "gestartet" werden , solange "der Roboterarm 1" "vorn" ist .*

```
AG !( rdy_plc & Arm1_ext=1.0 & (Robot_right) )
```

*/* >>>Satz 161<<< (einfaches Verbot - selbstbegrenzt) */*

*"die Drehung des Roboters nach links" darf niemals "gestartet" werden , solange "der Roboterarm 2" "vorn" ist .*

```
AG !( rdy_plc & Arm2_ext=1.0 & (Robot_left) )
```

*/* >>>Satz 162<<< (einfaches Verbot - selbstbegrenzt) */*

*"die Drehung des Roboters nach rechts" darf niemals "gestartet" werden , solange "der Roboterarm 2" "vorn" ist .*

```
AG !( rdy_plc & Arm2_ext=1.0 & (Robot_right) )
```

*/* >>>Satz 174<<< (einfaches Verbot - Zustand) */*

*"das Schließen der Presse" darf nicht gleichzeitig "gestartet" sein , wenn "der Roboter" "mit dem Arm 1 vor der Presse" ist und "der Roboterarm 1" ist "in der Presse" .*

```
AG !( rdy_plc & Robot_angle=-90 & Arm1_ext=0.65 &
(Press_upward) )
```

*/* >>>Satz 175<<< (einfaches Verbot - Zustand) */*

*"das Schließen der Presse" darf nicht gleichzeitig "gestartet" sein , wenn "der Roboter" "mit dem Arm 2 vor der Presse" ist und "der Roboterarm 2" ist "in der Presse" .*

```
AG !( rdy_plc & Robot_angle=0 & Arm2_ext=0.79 &
(Press_upward) )
```

## 5.4 Kommentare zur Fallstudie

Die dargestellten Ausführungen zur Fallstudie zeigen, dass es mit der Sicherheitsfachsprache möglich ist, sämtliche Anforderungen an ein Steuerungsprogramm natürlichsprachlich und dennoch formal darzustellen. Hierbei besteht darüber hinaus die Möglichkeit, eine Vielzahl unterschiedlicher Formulierungen für dieselbe Problemstellung zu verwenden. Die gewollte Redundanz der Formulierungen innerhalb einer Kategorie dient dazu, dem potentiellen Anwender eine Reihe von sprachlichen Varianten anzubieten, aus denen er sich diejenige auswählen kann, die ihm am Besten gefällt.

Die erstellten formalen Anforderungen können nun für eine Synthese des gewünschten Programms oder für eine Verifikation durch Model-Checking herangezogen werden. Bei letzterem Verfahren erhält der Nutzer zu jeder Analyseanfrage eine Antwort, ob diese spezielle Anforderung erfüllt wird oder nicht. Zusätzlich kann, je nach Typ der Analyseanfrage, ein Zeugniszustand bzw. ein Gegenbeispiel aufgezeigt werden. Meist wird durch den Model-Checker sogar der komplette Pfad bis zu diesem relevanten Zustand dargestellt.

Der Anwender ist bei der Verifikation oft negativ über die verhältnismäßig hohe Anzahl von Analyseergebnissen überrascht, die als falsch zurückgewiesen werden. Im ersten Moment wird die Ursache hierfür in einem fehlerhaften Systemmodell, und speziell einem fehlerhaften Programm vermutet. Das Erstaunen ist umso größer, wenn sich das Programm zuvor im praktischen Einsatz als scheinbar korrekt und sicher gezeigt hat.

Die Ursachen für eine relativ starke Fehlerquote beim Model-Checking liegen jedoch meist an anderer Stelle. Diese zeigen sich erst bei einem detaillierten statischen oder dynamischen Test des Programms bzw. der gründlichen Analyse des zum Fehlerzustand führenden Pfads. Eine wesentliche Ursache liegt in der Anforderung selbst. Viele Anfragen an das Verifikationswerkzeug werden zu allgemein oder zu oberflächlich gestellt, so dass bei der Analyse weitaus mehr Systemzustände analysiert werden müssen, als dem Nutzer bewusst ist. Grund hierfür ist die bereits dargestellte Eigenschaft natürlicher Sprache, dass man sich bei der Formulierung einer Anforderung auch auf vorherige Anforderungen beziehen kann. Bei einer formalen Darstellung darf diese elliptische Verkürzung jedoch nicht auftreten. Jede Aussage muss inhaltlich vollständig und abgeschlossen sein und muss im Besonderen alle Nebenbedingungen enthalten.

Durch Verfeinerung und Vervollständigung der Analyseanfrage erhält man schließlich auch eine inhaltlich korrekte Spezifikation.

# 6 Zusammenfassung und Ausblick

Mit der vorgestellten Sicherheitsfachsprache ist es möglich, die gewünschten Funktions- und Sicherheitseigenschaften eines Steuerungsprogramms verbal zu formulieren. Durch die Nutzung der natürlichen Sprache ist jetzt eine sehr elementare und auch für Nichtspezialisten verständliche Darstellungsform gegeben. Die SFS geht damit über die Möglichkeiten bisheriger Beschreibungsmittel hinaus, weil mit ihr einerseits eine Verständigung zwischen verschiedenen Projektpartnern möglich wird und mit der Kategorie der Verbote außerdem Anforderungen darstellbar werden, die mit bisher verfügbaren Beschreibungsmitteln nicht formulierbar waren.

Da die Formulierung von Programmanforderungen mit Hilfe der natürlichen Sprache aufgrund ihrer Mehrdeutigkeit zunächst sehr problematisch ist, wurde die Sicherheitsfachsprache gemäß den typischen steuerungstechnischen Zusammenhängen, Anforderungen und Formulierstilen definiert. Sie bietet durch die formale Vorgabe einer Syntax die Möglichkeit der Hinterlegung einer formalen Semantik, durch die Vorgabe fester Satzmuster und -strukturen werden klare und leicht lesbare Darstellungen erzeugt. Ziel der SFS ist es weiterhin, den Anwender durch eindeutige und leicht verständliche Formulierungen zu einer klaren Problemdefinition zu zwingen.

Als Grundmuster der Sätze dient das Konditional („*wenn ..., dann ...*“), weil dem Anwender das Denken in solchen Kausalketten (auf eine erfüllte Bedingung folgt eine Reaktion) entgegenkommt. Die Anforderungskategorien werden durch einen Modal- und einen Zeitparameter definiert. Der Modalparameter bestimmt dabei den Inhalt einer Anforderung, der Zeitparameter bestimmt deren zeitliche Reichweite. Insgesamt sind in der Sicherheitsfachsprache 18 Anforderungskategorien definiert, wobei innerhalb jeder Kategorie weitere sprachlich unterschiedliche, jedoch inhaltlich gleichwertige Satzmuster vorhanden sind. Die Vielfalt und Redundanz wurde bewusst gewählt, um dem Anwender verschiedene Möglichkeiten der Formulierung seiner Aufgabenstellung zur Verfügung zu stellen.

Für die Überprüfung der Handhabbarkeit wurden mehrere Fallstudien angefertigt, die sowohl die Umsetzung der SPS-Variablen in die elementaren Verbalphrasen als auch die Erstellung kompletter Anforderungssätze beinhalten. Eine dieser Fallstudien ist auszugsweise im Text sowie im Anhang dargestellt.

Durch einen Compiler werden die Anforderungen in Formeln der Temporalen Logik (hier CTL) umgewandelt. Die Anwendung der Sicherheitsfachsprache gibt durch ihre anwenderfreundliche Schnittstelle jedem Nutzer die Möglichkeit, mit den Darstellungs- und Beschreibungsmöglichkeiten der Temporalen Logik umzugehen. Die erzeugten CTL-Formeln können innerhalb der vorgestellten Projektstruktur für eine anschließende Verifikation des Steuerungsprogramms durch einen Model-Checker verwendet werden. Dieser liefert als Ergebnis der Überprüfung entweder eine Bestätigung für die Einhaltung der Spezifikation in allen modellierten Zuständen des Systems oder mögliche Gegenbeispiele, in denen eine oder mehrere Anforderungen nicht erfüllt werden. Die Aussagekraft des Verifikationsergebnisses hängt jedoch immer von der Qualität der gegebenen Voraussetzungen (Programm, Umgebungsmodell, Anforderungen) ab. Als Ergebnis einer erfolgreichen Verifikation kann man somit

von einem korrekten SPS-Programm bezüglich der gegebenen Voraussetzungen sprechen.

Für die Weiterentwicklung der vorgestellten Sicherheitsfachsprache existieren mehrere Anknüpfungspunkte. Diese liegen in der Vielfalt der Einsatzmöglichkeiten einer verbalen Spezifikationssprache begründet. Zwar wurde die SFS speziell für die Formulierung der Anforderungen an reaktive Systeme konzipiert, die Ausweitung der Nutzung auf andere Gebiete ist jedoch denkbar. Zu diesem Zweck bieten die formale Definition und die Struktur der Sicherheitsfachsprache die Möglichkeit einer einfachen Erweiterung der Syntax und die Einbindung neuer sprachlicher Konstruktionen. Hier ist vor allem die Erweiterung der Möglichkeiten zur Benutzung der Variablentypen „Integer" oder „Real" zu nennen.

Diese Erweiterungen bieten dann die Möglichkeit, andere Anwendungsbereiche zu erschließen. Als Beispiel sei hier die Fuzzylogik erwähnt, wo gerade die Formulierung von Anforderungen mit natürlichsprachlichen Mitteln eine große Rolle spielt.

Die Anwendung einer formalen Systembeschreibung für die Synthese von Steuerungsprogrammen wurde bereits ausführlich dargestellt. Auch hier bietet die Sicherheitsfachsprache genügend Möglichkeiten und mit der Kategorie „DEs2 – Einfache Forderung – Direkt" auch bereits eine geeignete Kategorie, um den gewünschten Ablauf eines Steuerungsprogramms vollständig zu beschreiben. Die Überführung von CTL-Formeln in ein SPS-Programm ist bereits heute Inhalt anderer Forschungsvorhaben.

Ein weiteres Feld der Weiterentwicklung ist die Portierung der Sicherheitsfachsprache auf andere Sprachen, wie z. B. Englisch. Die hierfür notwendigen Änderungen wären relativ einfach zu bewerkstelligen. Unter Beachtung des gegebenen Satzbaus der englischen Sprache sind die formale Syntax der SFS in den Ebenen 3, 4 und 5, sowie der PreLexer anzupassen.

Mit der Portierung auf weitere Sprachen erschließt sich umgehend ein weiteres Anwendungsfeld der Sicherheitsfachsprache. Der Vorgang der Überführung eines natürlichsprachlichen Satzes in eine CTL-Formel ist reversibel, allerdings müsste hierzu jeweils eine Standard-Formulierung pro Anforderungskategorie definiert werden, da keine eineindeutige Zuordnung eines natürlichsprachlichen Satzes zu einer CTL-Formel besteht. Es können also auch CTL-basierte Anforderungen in natürliche Sprache rückübersetzt werden, sofern sie die Syntax der SFS benutzen. Mit dieser Festlegung und den CTL-Formeln als formale Basis lassen sich Übersetzer zwischen verschiedenen Sprachen konstruieren, die eine in Deutsch formulierte Anforderung zunächst in eine CTL-Formel übersetzen und dann in einen natürlichsprachlichen englischen Satz überführen.

Abschließend ist der mögliche Ausbau des SFS-Editors zu nennen. Hierbei lassen sich vor allem durch eine geeignete hierarchische Strukturierung der Anforderungen, z. B. gemäß der in der Fallstudie gezeigten Zerlegung eines Produktionsauftrages, die notwendige Gliederung und Übersichtlichkeit verbessern.

Die vorliegende Arbeit hat gezeigt, dass eine natürlichsprachliche und gleichzeitig formale Anforderungsformulierung möglich ist. Denn nur durch eine konsequente Vereinfachung der Methoden und Werkzeuge für die Softwareerstellung wird es zu-

künftig möglich sein, die notwendige Sicherheit der uns umgebenden technischen Systeme zu gewährleisten.

# 7 Literatur

[Ahre00] Ahrens, W.; Felleisen, M.; Schnieder, E.; Chouikha, M.: „Formale Prozessbeschreibungen - gestern, heute und morgen“, Automatisierungstechnische Praxis 42 (2000), atp 9/2000

[Agui90] Aguilera, C.; Berry, D. M.: „The Use of Repeated Phrase Finder in Requirement Extraction”, Journal of Systems and Software, vol. 13, pp. 209-230, 1990

[Alag98] Alagar, V. S.; Periyasamy, K.: „Specification of Software Systems”, Springer, New York, 1998

[Alur96] Alur, R.; Henzinger, T. A.; Ho, P.-H.: „Automatic Symbolic Verification of Embedded Systems”, IEEE Transactions on Software Engineering 22:181-201, 1996

[Balz96] Balzert, H.: „Lehrbuch der Software-Technik – Software-Entwicklung“, Spektrum Akademischer Verlag GmbH Heidelberg Berlin Oxford, 1996

[Balz98] Balzert, H.: „Lehrbuch der Software-Technik – Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung“, Spektrum Akademischer Verlag GmbH Heidelberg Berlin, 1998

[Best95] Best, E.; Grahlmann, B.: „PEP - Programming Environment Based on Petri Nets, Documentation and User Guide”, Universität Hildesheim, Institut für Informatik, 1995

[Beus94] Beuschel, J.: „Prozesssteuerungssysteme“, Oldenbourg-Verlag München, 1994

[Bibe93] Bibel, W.: „Wissensrepräsentation und Inferenz – Eine grundlegende Einführung“, Vieweg-Verlag, Braunschweig Wiesbaden, 1993

[Bieg98] Biegert, U.: „Gefahrenanalyse auf der Basis von qualitativen Modellen“, Institut für Automatisierungs- und Softwaretechnik der Universität Stuttgart, Dechema Jahresbericht, 1998

[Birk00] Birkhofer, H.; Schulz, J.; Nötzke, D.: „Anforderungen nutzen - Umfassende Dokumentation und effizienter Zugriff auf Anforderungen durch XML-Technologie“, Zeitschrift für wirtschaftliche Fertigung 95 (2000), zwf 10/2000

[Bits00] Bitsch, F.; Gunzert, M.; „Formale Verifikation von Softwarespezifikationen in ASCET-SD und MATLAB“, Fachtagung Verteilte Automatisierung, 22.-23. März 2000, Magdeburg

[Bits02] Bitsch, F.; Göhner, P.: „Spezifikation von Sicherheitsanforderungen mit Safety-Patterns“, Tagungsband „Software Engineering in der industriellen Praxis“, VDI-Bericht-Nr. 1666, VDI-Verlag, Düsseldorf, S. 29-40, 2002

[Blac87] Black, W.J.: „Acquisition of conceptual data models from natural language descriptions”, in: Proc. of 3rd Conference of the european Chapter of computational linguistics, Copenhagen, Denmark, 1987

[Booc99] Booch, G. et al.: „Das UML-Benutzerhandbuch“, Addison-Wesley, Bonn, 1999

[Braa01] Braatz, A.; Große-Rhode, M.; Ehrig, H.; Westkämper, E.: „UML-basierte Software-Spezifikation und Entwicklungswerkzeuge für Systeme der Automatisierungstechnik“, Tagungsband EKA 2001, 7. Fachtagung Engineering komplexer Automatisierungssysteme, 25.-27. April 2001, Braunschweig

[Cane00] Canet, G.; Couffin, S.; Lesage, J.-J.; Petit, A.; Schnoebelen, Ph.: „Towards the automatic verification of PLC programs written in Instruction List”, IEEE International Conference of Systems, Man and Cybernetics, SMC 2000, pp. 2449-2454, Nashville, Tennessee (USA), 2000

[Canv97] Canver, E.; Gayen, J.-T.; Moik, A.: „Formale Spezifikation der Steuerungssoftware einer elektrisch ortsbedienten Weiche“, Automatisierungstechnische Praxis 39 (1997), atp 5/97

[Cauv91] Cauvet, C. et al.: „An expert System for requirement engeneering”, in: Proc. of Computer aides Information System Engineering, CAISE-91, Eds. R. Andersen et al., Trondheim, 1991

[Char92] Charwat, H. J.: „Lexikon der Mensch-Maschine-Kommunikation”, Oldenbourg-Verlag München, 1992

[Chou98a] Chouikha, M.; Schnieder, E.: „Modellbasierter Steuerungsentwurf mit Petri-Netzen“, GMA-Kongress Ludwigsburg, 18./19.06.1998, VDI Berichte Nr. 1397, 1998

[Chou98b] Chouikha, M.; Jahnsen, A.; Schnieder; E.: „Klassifikation und Bewertung von Beschreibungsmitteln für die Automatisierungstechnik“, Automatisierungstechnik 46 (1998), at 12/98, 1998

[Clar86] Clarke, E. M.; Emerson, E. A.; Sistla, A. P.: „Automatic Verification of finite-state concurrent systems using temporal logic specifications”, ACM Transactions on Programming Languages and Systems, Vol. 8, No. 2, April 1986

[Clar96] Clarke, E. M.; Wing, J. M.: „Formal Methods: State of the art and future directions”, Computer Science Department, Carnegie Mellon University, Pittburgh, ACM Press New York, 1996

[Dali95a] Dalianis, H.: „Aggregation in the NL-generator of the Visual and Natural Language specification tool", proc. 7th International Conference of the European Chapter of the Acciciation for Computer Linguistics, EACL'95, Dublin, Ireland, March 27-31, 1995

[Dali95b] Dalianis, H.: „Aggregation, Formal specification and Natural language Generation", proc. 1st International Workshop on the Applications of Natural Language to Data Base, NLDB'95, Versailles, France, June 28-29, 1995

[DeSm00] DeSmet, O.; Couffin, S.; Rossi, O.; Canet, G.; Lesage, J.-J.; Schnoebelen, Ph.; Papini, H.: "Safe Programming of PLC using formal verification methods", 4th international PLCopen conference on Industrial Control Programming, ICP2000, Utrecht (Netherlands), October 2000

[DGQ86] Deutsche Gesellschaft für Qualität: „Software-Qualitätssicherung – Aufgaben, Möglichkeiten, Lösungen", DGQ-ITG-Schrift Nr. 12-51, Beuth-Verlag, Berlin, 1986

[DGQ92] Deutsche Gesellschaft für Qualität: „Methoden und Verfahren der Software-Qualitätssicherung", DGQ-ITG-Schrift Nr. 12-52, Beuth-Verlag, Berlin, 1992

[DGQ98] Deutsche Gesellschaft für Qualität e.V.: „Zuverlässigkeit komplexer Systeme aus Hardware und Software", DGQ-Band 17 - 01, 1998

[DIN 31000] DIN VDE 31000: „Allgemeine Leitsätze für das sicherheitsgerechte Gestalten technischer Erzeugnisse", 1987

[DIN 19226] DIN 19226, „Leittechnik; Regelungstechnik und Steuerungstechnik", Teile 1 bis 6, 1994/97

[DIN 19250] DIN V 19250 (Vornorm): „Leittechnik, Grundlegende Sicherheitsbetrachtungen für MSR-Schutzeinrichtungen", 1994

[DIN 25419] DIN 25419: „Ereignisablaufanalyse: Verfahren, graphische Symbole und Auswertung", November 1985

[DIN 25424] DIN 25424: „Fehlerbaumanalyse, Methoden und Bildzeichen", 1981

[DIN 25448] DIN 25448: „Ausfalleffektanalyse (Fehler-Möglichkeits- und –Einfluß-Analyse)", Mai 1990

[DIN 40719] DIN 40719: „Schaltungsunterlagen", Teil 6: Regeln für Funktionspläne, 1992; Teil 11: Zeitablaufdiagramme, Schaltfolgediagramme, 1978

[DIN 44300] DIN 44300: „Informationsverarbeitung, Begriffe", Nov. 1988

[DIN 66255] DIN 66255: „Informationsverarbeitung, Programmiersprache PL/I“, 1980

[DIN 66285] DIN 66285: „Gütebedingungen und Prüfbestimmungen – Anwendersoftware – ISO/IEC CD 12199“

[DIN 66270] DIN 66270: „Informationstechnik – Bewerten von Softwaredokumenten – Qualitätsmerkmale“, 1998

[Dill94] Dillon, L. K.; Kutty, G.; Moser, L. E.; Melliar-Smith, P. M.; Ramakrishna, Y. S.: „A Graphical Interval Logic for Specifying Concurrent Systems”, ACM Transactions on Software Engineering and Methodology, Vol.3, No. 2, pp. 131-165, April 1994

[Dwye98] Dwyer, M. B.; Avrunin, G. S.; Corbett, J. C.: „Property specification patterns for finite-state verification”, Proc. 2nd Workshop on Formal Methods in Software Practice (FMSP-98), March, 1998

[Ehri85] Ehrig, H.; Mahr, B.: „Fundamentals of algebraic Specification”, Springer-Verlag, Berlin Heidelberg New York Tokyo, 1985

[Emer90] Emerson, E. A.: „Temporal and Modal Logic”; in: J. v. Leeuwen ed.: Handbook of theoretical computer science, vol. B, Elsivier Science Publishers B. V. Amsterdam, 1990

[Fein97] Feindt, E.-G.: „Entwurf und Simulation industrieller Steuerungen für den PC und die SPS“, Oldenbourg Verlag Wien, 1997

[Fish93] Fisher, M.; Owens, R.: „Executable Modal and Temporal Logics”, Springer-Verlag, IJCAI‘ 93 Workshop, Chambery, France 1993

[Flei97] Fleisch, W.: „Validierung von Entwurfsspezifikationen komponentenbasierter Software für verteilte, eingebettete Automatisierungssysteme mittels Simulation“, 4. Berichtskolloquium des GK PVS, Universität Stuttgart, 13. Juni 1997

[Flie86] Fliegner, J.: „Grammatik - verstehen und gebrauchen“, Scriptor-Verlag Frankfurt am Main, 1986

[Frey98] Frey, G.; Litz, L.: „Entwurf und formale Verifikation von Steuerungen mit interpretierten Petri-Netzen“, GMA-Kongress Ludwigsburg, 18./19.06.1998, VDI Berichte Nr. 1397, 1998

[Frey00] Frey, G; Litz, L.: „Formal methods in PLC programming”, IEEE Conference System, Man and Cybernetics SMC 2000, Nashville, TN, USA, Oct. 2000

[Frap01] Frappier, M.; Habrias, H. (Eds.): „Software Specification Methods”, Springer-Verlag London Berlin Heidelberg, 2001

[Futs89] Futschek, G.: „Programmentwicklung und Verifikation“, Springer-Verlag Wien, New York, 1989

[Glöe98] Glöe, G.; Jack, O.; Mehl, R.; Müllerburg, M.: „Zuverlässigkeit komplexer Systeme aus Hardware und Software“, Automatisierungstechnische Praxis 40 (1998), atp 1/98

[Gold94] Goldin, L.; Berry, D. M.: „AbstFinder, A Prototype Natural Language Text Abstraction Finder for Use in requirements Elicitation”, proc. 1st International Conference on Requirement Engineering, Colorado Springs, USA, 1994

[Grel93] Grell, D.; Hümbs, W.: „Safety First, Sicherheitstechnik im Computerzeitalter“, c't 4/93

[Habl98] Hablawetz, D.: „Applikationssoftware und Systemsicherheit in speicherprogrammierbaren Systemen“, Automatisierungstechnik 46 (1998), at 2/98

[Hala98] Halang, W. A.; Konakovsky, R.: „Sicherheitsgerichtete Software“, Automatisierungstechnik 46 (1998), at 2/98

[Hani96] Hanisch, H.-M.; Lüder, A.; Rausch, M.: „Automatische Codegenerierung für sicherheitsrelevante Steuerungen auf der Basis von Netzmodellen“, 3. Fachtagung Anlagen-, Arbeits- und Umweltsicherheit, Köthen, 7.-8. November 1996

[Hani98] Hanisch, H.-M.; Thieme, J.; Lüder, A.: „Steuerungssynthese auf der Basis von Netz-Condition/Event-Systemen“, GMA-Kongress Ludwigsburg, 18./19.06.1998, VDI Berichte Nr. 1397, 1998

[Hein97a] Heiner, M.; Meier. H.; Menzel, T.; Mertke, T.: „Petri-Netz-basierte Methoden zur sicherheitstechnischen Zertifizierung von SPS-Anwenderprogrammen“, BTU-Bericht Reihe Informatik I-19/1997, Cottbus, Dez. 1997, 26 Seiten, 1997

[Hein97b] Heiner, M., Menzel, T.: „Petri-Netz-Semantik für die SPS-Anwenderprogrammiersprache Anweisungsliste“, BTU-Bericht Reihe Informatik I-20/1997, Cottbus, 1997

[Hein99] Heiner, M., Menzel, T.: „Modellierung und Analyse von SPS-Anwenderprogrammen mit Petri-Netzen“, EKA '99, 6. Fachtagung, 247-265, Braunschweig, 1999

[Hein00a] Heiner, M., Menzel, T.: „Time-related modelling of PLC systems with time-less Petri nets“, Proc. WODES 2000, 275-282, 2000

[Hein00b] Heiner, M., Mertke, T., Menzel, T.: „Safety-Knight - Methoden und Werkzeuge zur Verifikation von SPS-Anwenderprogrammen“, SPS/IPC/DRIVES, Nürnberg, November 2000

[Hein01] Heiner, M.; Mertke, T.; Deussen, P.: „A safety-oriented Technical Language for the Requirement Specification in Control Engineering” (in deutsch), Computer Science Reports 09/01, May 2001, 65 pages

[Heri92] Hering, E.: „Software-Engineering”, Vieweg-Verlag Braunschweig, 1992

[Hofm00] Hofmann, P.; Fasolt, J.; Geretschläger, P.; Sakretz, R.; Wohlgemuth; F.: „Automotive UML - eine neue objektorientierte Entwicklungstechnik“, Elektronik Automotive 2000

[Hogr89] Hogrefe, D.: „Estelle, LOTOS und SDL, Standard- Spezifikationssprachen für verteilte Systeme“, Springer-Verlag Berlin, 1989

[Hölz96] Hölzlein, M.; Filkorn, Th.; Warkentin, P.: „Formale Verifikation von SPS-Programmen“, VDI/VDE-GMA-Kongreß Meß- und Automatisierungstechnik, Baden-Baden 1996, VDI-Bericht Nr. 1282, 1996

[Hölz98a] Hölzlein, M.; Filkorn, Th.; Warkentin, P.; Weiß, M.: „Erfahrungen mit der formalen Verifikation von Zustandsgraphen-Programmen“, GMA-Kongress Ludwigsburg 18./19.06.1998, VDI-Berichte 1397, 1998

[Hölz98b] Hölzlein M. ; Filkorn, Th. ; Warkentin, P. ; Weiß, M.: „Eine Verifikationskomponente für HiGraph“, GMA-Kongress Ludwigsburg, 18./19.06.1998, VDI Berichte Nr. 1397, 1998

[Hore89] Horebeek, I. v.; Lewi, J.: „Algebraic specifications in software engineering – an introduction”, Springer, Berlin Heidelberg, 1989

[Hube97] Huber, E.; Burgbacher, G.; Biegert, U.; Billmann, W.: „Qualitative Systemanalyse und computerunterstützte Gefahrenidentifikation (HAZOP)“, Wiley-VCH, Chemie-Ingenieur-Technik, 7/1997

[IEC1131] DIN IEC 61131-3 (IEC 1131-3): „Speicherprogrammierbare Steuerungen, Teil 3: Programmiersprachen“, 1993

[IEC1508] IEC 61508: „Functional Safety of Electrical/Electronic/ Programmable Electronic Safety Related Systems“, 2000

[Jörn96] Jörns, C.: „Spezifikation und Verifikation hierarchischer Steuerungen“, VDI/VDE-GMA-Kongreß Meß- und Automatisierungstechnik, Baden-Baden 1996, VDI-Bericht Nr. 1282, 1996

[Kasp94] Kaspers, W.; Küfner, H.-J.; Heinrich, B.; Vogt, W.: „Steuern-Regeln-Automatisieren“, Vieweg Fachbücher der Technik, 1994

[Kohr91] Kohring, A.: „Transparente Spezifikation komplexer Maschinen und Anlagen - Grundlage einer systematischen Entwicklung von SPS-Software“, Anwenderforum „Fortschrittliche Automatisierung mit SPS“, 4./5.12.1991 Bad-Soden VDI-Berichte 914, 1991

[Kohr93] Kohring, A.: „Systematisches Projektieren und Testen von Steuerungssoftware für Werkzeugmaschinen“, Dissertation an der RWTH Aachen, Verlag Shaker, Band 13/93, 1993

[Köst00] Köster, L.; Meyer, M.; Thomsen, T.: „Automatische Code-Generierung für Steuergeräte“, Elektronik 18/2000

[Koto97] Kotonya, G.; Sommerville, I.: „Requirements engineering – Processes and techniques”, Wiley & Sons Ltd, Chichester, England, 1997

[Kowa96] Kowalewski, S.: „Verifikation von Steuerungen mit Hilfe von Bedingung/Ereignis-Systemen“, VDI/VDE-GMA-Kongreß Meß- und Automatisierungstechnik, Baden-Baden 1996, VDI-Bericht Nr. 1282, 1996

[Kowa98] Kowalewski, S. ; Preußig, J.; Stursberg, O.; Treseler, H.: „Blockorientierte Modellierung und formale Verifikation von diskret gesteuerten kontinuierlichen Prozessen“, GMA-Kongress Ludwigsburg, 18./19.06.1998, VDI Berichte Nr. 1397, 1998

[Kowa01] Kowalewski, S.; Herrmann, P.; Engell, S.; Huuck, R.; Krumm, H.; Lakhnech, Y.; Lukoschus, B.; Treseler, H.: „Approaches to the formal verification of hybrid systems“, Automatisierungstechnik 49, 2001, at 2/2001

[Krau00] Krause, F.-L.; Heimann, R.: „Sprache zur Beschreibung von Produktentwicklungsprozessen“, Zeitschrift für wirtschaftlichen Fabrikbetrieb 95 (2000), zwf 6/2000

[Krög87] Kröger, F.: „Temporal Logics of Programs”, Monographs on Theoretical Computer Science, Springer-Verlag, 1987

[Krüc90] Krückeberg, F.; Spaniol, O.: „Lexikon Informatik und Kommunikationstechnik“, VDI-Verlag Düsseldorf 1990

[Kürs93] Kürschner, W.: „Grammatisches Kompendium“, Francke-Verlag Tübingen, 1993

[Lamp99] Lamperiere-Couffin, S.; Rossi, O.; Roussel, J.-M.; Lesage, J.-J.: „Formal validation of PLC programs: a survey”, ECC '99: European Control Conference 1999, Karlsruhe, 31. August - 3. September 1999

[Lamp00] Lamperiere-Couffin, S.; Lesage, J.J.: „Formal Verification of the sequential part of PLC Programs”, 5th workshop on Discrete Event Systems, WODES 2000, pp. 247-254, Gent (Belgium), August 2000

[Lemm95] Lemmer, K.; Ober, B.; Schnieder, E.: „Model-based Programming and Diagnosis for programmable logical Controllers“, IEEE International Conference on Systems, Man and Cybernetics (SMC95), Vancouver, Canada, 22.-25.10.1995

[Leve95] Leveson, N. G.: „Safeware - System safety and computers”, Addison-Wesley Publishing Company, 1995

[Lewe95] Lewerentz, C.; Lindner, Th. (Eds.): „Formal Development of Reactive Systems - Case study Production Cell", Springer-Verlag Berlin Heidelberg New York, 1995

[Lewe97] Lewerentz, C.; Rust, H.: „Zur Bedeutung von Spezifikationen in verschiedenen Teilaufgaben der Entwicklung kundenspezifischer Software“, Informatik-Berichte I-21/1997, Brandenburgische Technische Universität Cottbus, 1997

[Lind93] Lindermeier, R.: „Softwarequalität und Softwareprüfung“, Oldenbourg-Verlag GmbH, München 1993

[Litz98] Litz, L.: „Grundlagen der sicherheitsgerichteten Automatisierungstechnik“, Automatisierungstechnik 46 (1998), at 2/98

[Litz99] Litz, L.; Frey, G.: „Methoden und Werkzeuge zum industriellen Steuerungsentwurf - Historie, Stand, Ausblick“, Automatisierungstechnik 47 (1999), at 4/99

[Mann92] Manna, Z.; Pnueli, A.: „The temporal logic of reactive and concurrent systems – specification", Springer-Verlag, New York, 1992

[Mann95] Manna, Z.; Pnueli, A.: „Temporal Verification of Reactive Systems – Safety", Springer-Verlag, New York, 1995

[McMi92] McMillan, K. L.: „The SMV System", Techn. Report, Carnegie-Mellon Univ. 1992

[Meff99] Meffert, K.; Reinert, D.: „Sicherheitsreport, Mikroprozessoren in sicherheitskritischen Anwendungen“, Teile 1 und 2, Elektronik 4 + 6 /99

[Meie98] Meier, H.; Mertke, T.: „Verifikation zertifizierungspflichtiger Steuerungssoftware“, Zeitschrift für wirtschaftlichen Fabrikbetrieb, ZwF 6/98, S. 247 – 250, 1998

[Meie99] Meier, H.; Mertke, T.: „Eine Sicherheitsfachsprache zur Formulierung steuerungstechnischer Anforderungen“, Zeitschrift für wirtschaftlichen Fabrikbetrieb, ZwF 11/99, S. 660 – 664, 1999

[Mert01] Mertke, T.; Deussen, P.; Heiner, M.: „Eine anwenderorientierte Sicherheitsfachsprache zur Verifikation von Steuerungsprogrammen“, Tagungsband EKA 2001, 7. Fachtagung Engineering komplexer Automatisierungssysteme, S. 297 – 310, 25.-27. April 2001, Braunschweig

[Mewe95] Mewes, J.; Müller, U.: „Software frühzeitig aus Kundensicht prüfen“, Qualität und Zuverlässigkeit 40 (1995), 4/95

[Mont00] Montenegro, S.: „Doppelt gedacht hält besser - Fehlertoleranz gegen Entwicklungsfehler“, Elektronik 8/2000

[Moon91] Moon, I., Powers, G. J., Burch, J. R., Clarce, E. M.: „An automatic verification method using temporal logic for sequential chemical process control systems”, AIChE Journal, vol. 38, p.67, 1991

[Mord83] Mordechai, B.-A.; Pnueli, A.; Manna, Z.: „The temporal logic of branching time“, Acta Informatica, Springer-Verlag, 1983

[Oest98] Oesterreich, B.: „Objektorientierte Softwareentwicklung, Analyse und Design mit der Unified Modeling Language“, Oldenbourg Verlag München Wien, 1998

[Part98] Partsch, H.: „Requirements-Engineering systematisch”, Springer-Verlag, Berlin, 1998

[Polk92] Polke, M.: „Prozessleittechnik“, Oldenbourg-Verlag GmbH München, 1992

[Reck91] Reck, M.: „Methoden und Beschreibungsmittel für die Programmentwicklung“, Schriftenreihe: Integrierte Datenverarbeitung in der Praxis, Band 47, Forkel-Verlag Wiesbaden, 1991

[Rein96] Reinhard, H.: „Automatisierungstechnik – Theoretische und gerätetechnische Grundlagen, SPS“, Springer-Verlag Berlin Heidelberg New York, 1996

[Rein99] Reinert, D.; Schaefer, M.; Börner, Th.: „Regeln für den Entwurf und die Programmierung sicherheitsbezogener Software“, Automatisierungstechnische Praxis 41 (1999), atp 6/99

[Reis85] Reisig, W.: „Systementwurf mit Netzen“, Springer-Verlag, Berlin Heidelberg New York Tokyo, 1985

[Ried83] Riedewald, G.; Maluszynski, J.; Dembinski, P.: „Formale Beschreibung von Programmiersprachen“, Akademie-Verlag, Berlin, 1983

[Ross00a] Rossi, O.; deSmet, O.; Couffin, S.; Lesage, J.-J.; Papini, H.; Guennec, H.: „Formal Verification: A Tool to improve the safety of control systems”, $4^{th}$ Symposium on Fault Detection, Safety and Supervision of Technical Processes, SafeProcess 2000, pp. 885-890, IFAC Budapest (Hungary), June 2000

[Ross00b] Rossi, O.; Schnoebelen, P.: „Formal Modeling of Times Function Blocks for the automatic Verification of Ladder Diagram Programs”, $4^{th}$ International Conference Automation of Mixed Processes, ADPM 2000, pp. 177-182, Dortmund, September 2000

[Rupp96] Ruppen, P.: „Einstieg in die formale Logik“, Lang-Verlag Bern, 1996

[Rust94] Rust, H.: „Zuverlässigkeit und Verantwortung, Die Ausfallsicherheit von Programmen“, DUD-Fachbeiträge, Nr. 21, Vieweg-Verlag, 1994

[Sche00] Schedl, P.: „Nahtlos von der Spezifikation zum Code - Modellbasierte Entwicklung von Kfz-Steuergeräten“, Elektronik Automotive 2000

[Schr92] Schrüfer, E.: „VDI-Lexikon Meß- und Automatisierungstechnik”, Springer-Verlag Berlin Heidelberg, 1992

[Somm97] Sommerville, I.: „Software Engineering”, Addison-Wesley, Harlow England, 1997

[Star92] Starke, P. H.: „INA - Integrated Net Analyser Manual“, Berlin, 1992

[Stet87] Stetter, F.: „Softwaretechnologie: Eine Einführung“, Reihe Informatik, Band 33, Wissenschaftsverlag Mannheim, Wien, Zürich, 1987

[Stöl98] Stölzl, S.; Isermann, R.; Rieth, P.: Nell, J.: „Methodik zur Erarbeitung von Überwachungsverfahren für sicherheitskritische verteilte Echtzeitsysteme“, GMA-Kongress Ludwigsburg, 18./19.06.1998, VDI Berichte Nr. 1397 (1998)

[Stru91] Struß, P.: „Wissensrepräsentation“, Oldenbourg-Verlag München, 1991

[Tres00a] Treseler, H.; Bauer, N.; Kowalewski, S.: „Verification of IL programs with an explicit model of their PLC execution”, 5th Workshop on Discrete Event Systems, WODES, August 2000

[Tres00b] Treseler, H.; Bauer, N.; Kowalewski, S.: „Model-Checking von AWL Programmen“, Fachtagung Verteilte Automatisierung 22./23. März 2000, Magdeburg

[Varp95] Varpaaniemi, K.; Halme, J.; Hiekkanen, K.; Pyssyslao, T.: „PROD Reference Manual”, Helsinki Univ. of Technology, Digital Systems Laboratory, Series B: Techn. Report no. 13, Espoo, 1995

[VDI2180] VDI/VDE 2180: „Sicherung von Anlagen der Verfahrenstechnik mit Mitteln der Prozessleittechnik (PLT)“, Blatt 1 bis 5, 1998

[VDI3542] VDI/VDE 3542: „Sicherheitstechnische Begriffe für Automatisierungssysteme“, Blatt 1 bis 4, Oktober 2000

[VDI3694] VDI-Richtlinien, VDI/VDE 3694: „Lastenheft/Pflichtenheft für den Einsatz von Automatisierungssystemen”, April 1991

[Wehr96] Wehrheim, H.: „Specifying reactive Systems with action dependencies”, PhD thesis, University of Hildesheim, 1996

[West01] Westkämper, E.; Braatz, A.: „Eine Methode zur objektorientierten Softwarespezifikation von zentralen Automatisierungssystemen mit der Unified Modeling Language (UML)“ Automatisierungstechnik 49 (2001), at 5/2001

[Xu01] Xu, L.; Bender, K.: „Wiederverwendungsorientierter Aufbau einer Modulbibliothek für die Maschinensimulation“, Tagungsband EKA 2001, 7. Fachtagung Engineering komplexer Automatisierungs-systeme, 25.-27. April 2001, Braunschweig, 2001

WWW:

Links zum Thema „Prüfen und Testen von Software“:

www.informatik.uni-koeln.de/winfo/prof.mellis/tav/links_inhalt.html

Links zum Thema „Verifikation“:

www.cs.tu-bs.de/ips/huhn/lehre/verification_links.html

Links zum Thema „Formale Methoden“

www.comlab.ox.ac.uk/archive/formal-methods.html – Formale Methoden, Tools

www-comp.mpce.mq.edu.au/~didar/seweb/tools.html – Requirement engineering, Tools and techniques

www.cs.toronto.edu/~chechik/courses00/csc2108/

www.science.unitn.it/~masini/FORMAL/local-formal.html

Links zum Thema „Model-Checking”, Tools

www.cc.ioc.ee/~juhan/fmmf/modchk-links.html

www.fi.muni.cz/paradise/acp_tools/acsedit/cgi.cs.iso-8859-2 -Verifikationswerkzeuge

www.cs.indiana.edu/formal-methods-education/tools – Formale Methoden, Tools

# Anhang

## A.1 Begriffsbestimmungen

Bei der Überführung formaler Methoden in die praktische Anwendung ergeben sich häufig Verständigungsschwierigkeiten, weil bestimmte Begriffe aus dem Kontext der Informatik oder der Automatisierungstechnik heraus unterschiedlich verwendet werden. Deshalb sollen an dieser Stelle einige relevante Begriffe und deren Verwendung innerhalb dieser Arbeit erläutert werden.

Zunächst sollen einige Begriffe der Systemtheorie dargestellt werden. Dieser Bereich ist erwiesenermaßen sehr vielschichtig, die verwendeten Begriffe sind oftmals durch die Betrachtungsweise der jeweiligen Wissenschaftsrichtung geprägt. Die folgende Ausführung orientiert sich am Bereich der Steuerungs-, Regelungs- und Leittechnik [DIN 19226, 1994].

Ein ***System*** (engl. system) ist eine in einem betrachteten Zusammenhang gegebene Anordnung von Komponenten, die in einem bestimmten inneren Zusammenhang zueinander stehen. Diese Anordnung wird aufgrund bestimmter Vorgaben gegenüber ihrer Umgebung abgegrenzt. Ein System hat eine durch seine *Struktur* gegebene Ordnung und eine durch seine Dynamik und Kausalität gegebene Funktion. Die ***Struktur*** (engl. structure) ist die Gesamtheit der Beziehungen zwischen den Komponenten des Systems. Diese Beziehungen können räumlich, zeitlich, begrifflich oder funktionell sein, sie beschreiben, welche Komponenten wie miteinander verknüpft sind, d. h. die statischen Eigenschaften des Systems. Die ***Systemparameter*** eines Systems sind *Größen*, deren Werte das *Verhalten* des Systems bei gegebener *Struktur* kennzeichnen. Eine ***Größe*** beschreibt die Eigenschaft eines Vorgangs oder Körpers, die einer qualitativen Identifizierung und einer quantitativen Bestimmung zugänglich ist. Der Wert einer Größe ist das Ergebnis ihrer quantitativen Bestimmung, das als Produkt aus Zahlenwert und Einheit angegeben wird. Der ***Zustand*** eines Systems ist seine Beschaffenheit im Augenblick der Betrachtung. Das ***Verhalten*** (engl. behaviour) eines Systems beschreibt die Gesamtheit der zeitlichen und kausalen Änderungen von Merkmalen des Systems. Es bezeichnet die Form des Agierens und Reagierens des Systems infolge bestimmter Bedingungen, es beschreibt somit die Dynamik des Systems. Besteht eine feste Beziehung zwischen dem Reiz und der zugehörigen Reaktion, so spricht man von deterministischem Verhalten. Durch das ***Verhalten*** des Systems in bestimmten Situationen werden seine ***Eigenschaften*** (engl. characteristics) bestimmt.

Ein ***Prozess*** (engl. process) ist die Gesamtheit von aufeinander einwirkenden Vorgängen in einem *System*, durch die Materie, Energie oder Information umgeformt, transportiert oder gespeichert wird. Ein ***Modell*** (engl. model) ist eine Abbildung eines *Systems* oder *Prozesses* in ein anderes begriffliches oder gegenständliches System, das aufgrund der Anwendung bekannter Gesetzmäßigkeiten, einer Identifikation oder auch getroffener Annahmen gewonnen wird und das System oder den Prozess bezüglich ausgewählter Fragestellungen hinreichend genau abbildet. Es stellt die als wesentlich erachteten Eigenschaften eines Systems in einer zweckmäßigen Form dar.

Eine ***Methode*** (engl. method) bestimmt und beschreibt die planmäßige Vorgehensweise, um ein bestimmtes (wissenschaftliches oder praktisches) Ziel zu erreichen [Chou98b]. Durch sie wird die Art und die Anwendung der einzusetzenden Mittel, sowie die Reihenfolge der durchzuführenden Aktivitäten festgelegt. In Verbindung mit einem ***Werkzeug*** (Tool, rechnerunterstütztes Hilfsmittel) wird aus einer Methode ein ***Verfahren*** (engl. procedure).

Ein ***Algorithmus*** (engl. algorithm) [DIN 19226/1] ist eine vollständig festgelegte endliche Folge von Vorschriften, nach denen aus zulässigen Eingangsgrößen des Systems die gewünschten Ausgangsgrößen erzeugt werden.

Ein ***Programm*** (engl. program) löst ein bestimmtes Problem anhand einer Folge logisch aufeinander abgestimmter Anweisungen [DIN 44300]. Die Formulierung erfolgt in einer *Programmiersprache*, die verwendeten Anweisungen werden dann Befehle genannt. Voraussetzung für ein Programm ist die Berechenbarkeit der Lösung.

Eine ***Sprache*** (engl. language) ist ein Formalismus zur Darstellung von Sachverhalten durch Folgen von Zeichen, deren Reihenfolge durch die Syntax der Sprache festgelegt ist [Krüc90].

Die ***Semiotik*** der Sprache beschreibt die definierte Menge von Zeichen bzw. Symbolen.

Die ***Syntax*** (engl. syntax) beschreibt mit Hilfe von Regeln für zulässige Zeichenkombinationen die Anordnung der Wörter für formal korrekte Sätze einer Sprache. Die ***Semantik*** beschreibt deren Bedeutung. [Ahre00]

Eine ***Programmiersprache*** (engl. programming language) ist eine künstliche Sprache, mit der *Programme* so formuliert werden können, dass Rechner in der Lage sind, sie abzuarbeiten. Sie zeichnen sich durch eine strikte Zweckgebundenheit sowie eine klare und einfache Syntax aus. Sie sind ein semi-formales Beschreibungs- bzw. Ausdrucksmittel für *Algorithmen*.

**Steuerungstechnik** [DIN 19226]

Unter ***Steuern*** bzw. einer ***Steuerung*** (engl. control) versteht man den Vorgang in einem *System*, bei dem eine oder mehrere *Größen* als Eingangsgrößen andere Größen als Ausgangsgrößen aufgrund der dem System eigentümlichen Gesetzmäßigkeiten beeinflussen. Hierzu zählen alle Maßnahmen zur gerichteten und planmäßigen Beeinflussung von Abläufen und Prozessen, um ein vorgegebenes Sollziel zu erreichen [Schr92]. Die Berechnungsvorschrift, nach der das Steuern stattfindet, wird als ***Steueralgorithmus*** bezeichnet.

Das ***Steuergerät*** bzw. das ***Automatisierungsgerät*** ist das Betriebsmittel, das die Steuerungsaufgaben bearbeitet [Schr92]. Es stellt die technische Realisierung der Steuereinrichtung dar. Oftmals wird auch der Begriff der *Steuerung* synonym verwendet, ein typisches Automatisierungsgerät ist eine ***Speicherprogrammierbare Steuerung*** (SPS, engl. PLC).

***Anforderungen*** (engl. requirements) sind qualitative und/oder quantitative Festlegungen der *Eigenschaften* eines Produkts.

Die ***Spezifikation*** (engl. specification) ist ein Dokument, das die *Anforderungen* bzw. Merkmale eines Produkts oder einer Dienstleistung festlegt [DIN 66255]. Sie enthält Aussagen zu Qualitätsanforderungen, Gebrauchstauglichkeit, Sicherheit, Abmessungen usw. des Produkts. Die Spezifikation stellt eine generalisierte Bedarfsanforderung dar [DGQ98], [Glöe98]. Eine *Spezifikation* eines *Programms* ist eine Formulierung dessen, was ein Programm tun soll. Man kann hierbei eine externe und eine interne Spezifikation unterscheiden. Die externe Spezifikation ist eine informelle (meist verbale) Beschreibung des Programmverhaltens, also ein Katalog von Anforderungen, die das Programm erfüllen soll. Aus der externen Spezifikation versucht man eine interne formale Spezifikation abzuleiten. Diese enthält die formale Definition der Aufgaben des Programms (vgl. auch Lastenheft/ Pflichtenheft).

Zur Erstellung einer Spezifikation werden ***Beschreibungsmittel*** (engl. means of description / representation) verwendet. Diese dienen zur Beschreibung und Darstellung der Struktur und/oder des Verhaltens eines Systems und stellen bestimmte Sachverhalte in grafischer Form zur visuellen Wahrnehmung und Speicherung dar (z. B. durch alphanumerische Zeichen, Symbole, Diagramme oder sonstige grafische Darstellungselemente). Je nach Definitionsgrad erfolgt eine Unterscheidung in informale, semi-formale oder formale Beschreibungsmittel. Formale Beschreibungsmittel haben eine mathematische Basis und eine definierte vollständige Syntax, semi-formale Beschreibungsmittel besitzen zwar eine definierte vollständige Syntax, aber keine mathematische Basis. Informale Beschreibungsmittel haben grundsätzlich keine vollständige Ausprägung dieser Merkmale [Chou98b]

***Formale Spezifikationstechniken*** zeichnen sich dadurch aus, dass mathematische Methoden zur Abschätzung wesentlicher Eigenschaften der zu beschreibenden Systeme anwendbar sind [Krüc90]. Für verschiedene Arten von Systemen haben sich Spezifikationstechniken entwickelt, die auf die jeweilige Problemstellung zugeschnitten sind. Einfache Beispiele hierfür sind elektrische Schaltbilder und Baupläne. Die Syntaxen von Spezifikationstechniken sind teilweise standardisiert (z. B. Protokollspezifikationen: LOTOS, Estelle, SDL, ASN.1) oder auch nichtstandardisiert (Petri-Netze).

Unter ***Programmsynthese*** versteht man das automatische Erstellen eines *Programms*, das ein vorgegebenes Problem lösen soll. Grundlage der Synthese ist eine formale Problem- bzw. Programmspezifikation, außerdem muss bekannt sein, in welcher Zielsprache das Programm erstellt werden soll.

Ein ***Programmtest*** beinhaltet die Überprüfung der Einhaltung der *Programmspezifikation* durch ein probeweises Ausführen des Programms. Dazu werden Testfälle bestimmt, die erwarteten Ergebnisse werden mit den Reaktionen des Programms verglichen. Je nach Kenntnis von der realen Implementierung des Programms unterscheidet man den black-box-Test und den white-box-Test. Programmtests sind immer partiell und zeigen nur das Vorhandensein von Fehlern, nicht aber deren Abwesenheit.

***Verifikation*** bedeutet, dass die Soll-Eigenschaften eines Programms durch einen exakten Beweis nachgewiesen werden. Die Führung des Beweises erfolgt durch lo-

gische Schlüsse auf der Grundlage von Axiomen und bereits bewiesenen Aussagen. Hierfür ist eine Formalisierung der Soll-Eigenschaften notwendig. Die Verifikation stellt die Korrektheit von Programmen gegenüber ihrer Spezifikation sicher. [VDI/VDE 3542]

In Erweiterung zur *Verifikation* wird bei einer ***Validierung*** auch die Richtigkeit der Spezifikation gegenüber der ursprünglichen Aufgabenstellung überprüft. Eine Validierung erfolgt hauptsächlich durch experimentelle Verfahren und Maßnahmen zur Überprüfung der Systemeigenschaften (z. B. durch Simulationen und Tests).

Ein ***Ausfall*** (engl. failure, defect) kennzeichnet die Funktionsunfähigkeit einer Komponente eines Systems [VDI/VDE 3542] [[Char92] /[Krüc90]]. Ausfälle lassen sich durch unterschiedliche Kriterien unterscheiden: Ursache (zufällig, deterministisch), Umfang (Totalausfall, Teilausfall), Geschwindigkeit (Sprungausfall, Driftausfall), Erkennbarkeit, Wirkung, Richtung u. a.

Ein ***Fehler*** (engl. error) [DIN 19226] ist eine unzulässige Abweichung vom aufgabengemäßen Verhalten des Systems oder einer seiner Komponenten (Nichterfüllung einer festgelegten Forderung). Im Zusammenhang mit einem Steuerungsgerät lassen sich aktive und passive Fehler unterscheiden. Bei einem aktiven Fehler werden Steuerungsfunktionen ausgelöst, ohne dass die programmgemäß festgelegten Bedingungen erfüllt sind. Von einem passiven Fehler spricht man, wenn Steuerungsfunktionen blockiert sind, obwohl alle programmgemäß festgelegten Bedingungen erfüllt sind [Litz98].

Ein ***Spezifikationsfehler*** liegt vor, wenn bereits bei der Spezifikation der Soll-Eigenschaften eines Systems Fehler auftreten.

***Sicherheit*** [DIN 31004] ist „eine Sachlage, bei der das ***Risiko*** kleiner als das Grenzrisiko ist". Das ***Grenzrisiko*** ist dabei „das größte noch vertretbare anlagenspezifische Risiko eines bestimmten technischen Vorgangs oder Zustands". Für die daraus folgende Notwendigkeit der Definition des Grenzrisikos führt die [DIN 31004] aus: „Es wird in der Regel durch sicherheitstechnische Festlegungen abgegrenzt, die den Schutzzielen des Gesetzgebers folgend nach der unter Sachverständigen vorherrschenden Auffassung getroffen werden." Es gibt also keine absolute Maßangabe für das Risiko. Um eine gewisse Quantifizierung des Risikos zu ermöglichen, haben sich verschiedene Ansätze herausgeprägt, die das Risiko in Klassen oder Levels einteilen. So legt die [DIN 19250] acht Anforderungsklassen fest, wobei höhere Anforderungsklassen eine höhere Zuverlässigkeit der eingesetzten Systemkomponenten verlangen.

Die ***Zuverlässigkeit*** [VDI/VDE 2180] (engl. reliability) eines Systems ist deren Fähigkeit, eine vorgegebene Funktion innerhalb vorgegebener Grenzen und für eine bestimmte Zeit zu erfüllen. Die ***Verfügbarkeit*** (engl. availability) ist eine Maßzahl für die Zuverlässigkeit, die kennzeichnet die Wahrscheinlichkeit, dass ein betrachtetes System zu einem bestimmten Zeitpunkt funktionsfähig ist.

Weiterführende zusammenhängende Darstellungen und Erläuterungen der genannten Begriffe können [Rust94], [Litz98], [Glöe98] und [DGQ98] entnommen werden.

## A.2 Formale Definition der Sicherheitsfachsprache

Die Darstellung der formalen Syntax der Sicherheitsfachsprache erfolgt in Backus-Naur-Form (siehe [Ried83]). Zur Verbesserung der Übersichtlichkeit der Darstellung erfolgt die Definition in verschiedenen Abstraktionsebenen.

### Ebene 0 - Definition der Satzliste

```
 <satzliste> ::=   <statement>
                 | <satzliste> <statement>
```

### Ebene 1 - Definition des Modalparameters

```
 <statement> ::=   <comment>
                 | <DEs>           (* einfache Forderung *)
                 | <DEe>           (* erweiterte Forderung *)
                 | <POe>           (* erweiterte Möglichkeit *)
                 | <PRs>           (* einfaches Verbot *)
```

### Ebene 2 - Definition des Zeitparameters

```
       <DEs> ::=   <DEs1>          (* Zustand *)
                 | <DEs2>          (* direkt *)
                 | <DEs3>          (* selbstbegrenzt *)
                 | <DEs4>          (* fremdbegrenzt *)
                 | <DEs5>          (* unbegrenzt *)

       <DEe> ::=   <DEe1>
                 | <DEe2>
                 | <DEe3>
                 | <DEe4>
                 | <DEe5>

       <POe> ::=   <POe1>
                 | <POe3>
                 | <POe4>
                 | <POe5>

       <PRs> ::=   <PRs1>
                 | <PRs3>
                 | <PRs4>
                 | <PRs5>

   <comment> ::= undefined
```

**Ebene 3 - Aufteilung in Satzvarianten, ab hier beruhen die Anforderungssätze auf gleiche CTL-Formeln**

```
                            (* ..._pl – Pluralkonstruktion *)
             (*..._for_... – forward (Vorwärtskonstruktion) *)
             (*..._back_... – backward (Rückwärtskonstruktion)
                                                           *)

<DEs1>  ::=    „Wenn“ <pres_cond_pl> „, dann“ <DE1_for_pl> „.“
             | <DEs1_back_pl> „, wenn“ <pres_cond_pl> „.“

<DEs2>  ::=    „Wenn“' <pres_cond_pl> „, dann“ <DE2_for_pl> „.“
             | <DEs2_back_pl> „, wenn“' <pres_cond_pl> „.“

<DEs3>  ::=    „Solange“ <pres_cond_pl> „,“' <DE3_for_pl> „.“
             | <DEs3_back_pl> „, solange“ <pres_cond_pl> „.“

<DEs4>  ::=    „Wenn irgendwann“ <past_cond_pl> „, dann“
             <DE3_for_pl>
               „, bevor“' <pres_cond_pl> „.“
             | „Nachdem“' <past_cond_pl> „,“ <DE3_for_pl>
               „, bevor“ <pres_cond_pl> „.“

<DEs5>  ::=    „Wenn irgendwann“' <past_cond_pl> „, dann“
             <DE3_for_pl> „.“
             | „Nachdem“ <past_cond_pl> „,“ <DE3_for_pl> „.“
             | <DEs3_back_pl> „, wenn irgendwann“ <past_cond_pl> „
             .“
             | <DEs3_back_pl> „, wenn vorher“ <past_cond_pl> „.“


<DEe1>  ::=    „Nur wenn“ <pres_cond_pl> „, dann“ <DE1_for_pl> „.“
             | <DEe1_back_pl> „, wenn gleichzeitig“ <pres_cond_pl> „
             .“

<DEe2>  ::=    „Nur wenn“ <pres_cond_pl> „, dann“ <DE2_for_pl> „.“
             | <DEe2_back_pl> „, wenn“ <pres_cond_pl> „.“

<DEe3>  ::=    „Nur solange“ <pres_cond_pl> „,“ <DE3_for_pl> „.“'
             | <DEe3_back_pl> „, solange gleichzeitig“ <pres_cond_pl>
             „.“

<DEe4>  ::=    „Nur wenn irgendwann“ <past_cond_pl> „, dann“'
             <DE3_for_pl>
               „, bevor“' <pres_cond_pl> „.“
             | „ Nur nachdem“ <past_cond_pl> „,“ <DE3_for_pl>
               „, bevor“ <pres_cond_pl> „.“
```

<DEe5> ::= „Nur wenn irgendwann“ <past_cond_pl> „ , dann“' <DE3_for_pl> „ .“
| „Nur nachdem“' <past_cond_pl> „ ,“ <DE3_for_pl> „ .“
| <DEe3_back_pl> „ , wenn irgendwann“ <past_cond_pl> „ .“
| <DEe3_back_pl> „ , wenn vorher“ <past_cond_pl> „ .“

<POe1> ::= „Nur wenn“ <pres_cond_pl> „ , dann“' <PO1_for_pl> „ .“
| <POe1_back_pl> „ , wenn gleichzeitig“ <pres_cond_pl> „ .“

<POe3> ::= „Nur solange“ <pres_cond_pl> „ ,“ <PO3_for_pl> „ .“
| <POe3_back_pl> „ , solange gleichzeitig“ <pres_cond_pl> „ .“

<POe4> ::= „Nur wenn irgendwann“ <past_cond_pl> „ , dann“ <PO3_for_pl>
„ , bis“ <pres_cond_pl> „ .“
| „Nur nachdem“ <past_cond_pl> „ , dann“ <PO3_for_pl>
„ , bis“ <pres_cond_pl> „ .“

<POe5> ::= „Nur wenn irgendwann“ <past_cond_pl> „ , dann“ <PO3_for_pl> „ .“
| „Nur nachdem“ <past_cond_pl> „ ,“ <PO3_for_pl> „ .“
| <POe3_back_pl> „ , wenn irgendwann“ <past_cond_pl> „ .“
| <POe3_back_pl> „ , wenn vorher“ <past_cond_pl> „ .“

<PRs1> ::= „Wenn“ <pres_cond_pl> „ , dann“ <PR1_for_pl> „ .“
| <PRs1_back_pl> „ , wenn“ <pres_cond_pl> „ .“

<PRs3> ::= „Solange“ <pres_cond_pl> „ ,“ <PR3_for_pl> „ .“
| <PRs3_back_pl> „ , solange“' <pres_cond_pl> „ .“

<PRs4> ::= „Wenn irgendwann“ <past_cond_pl> „ , dann“ <PR3_for_pl>
„ , bis“ <pres_cond_pl> „ .“
| „Nachdem“ <past_cond_pl> „ ,“ <PR3_for_pl>
„ , bis“ <pres_cond_pl> „ .“

<PRs5> ::= „Wenn irgendwann“ <past_cond_pl> „ , dann“' <PR3_for_pl> „ .“
| „Nachdem“ <past_cond_pl> „ ,“ <PR3_for_pl> „ .“
| <PRs3_back_pl> „ , wenn irgendwann“ <past_cond_pl> „ .“
| <PRs3_back_pl> „ , wenn vorher“ <past_cond_pl> „ .“

## Ebene 4 - Definition der konjunktiven Verknüpfungen

```
                                                  (* ..._sg – Singular *)

 <pres_cond_pl> ::=   <pres_cond_sg>
                    | <pres_cond_sg> „und“ <pres_cond_pl>

 <past_cond_pl> ::=   <past_cond_sg>
                    | <past_cond_sg> „und“ <past_cond_pl>

   <DE1_for_pl> ::=   <DE1_for_sg>
                    | <DE1_for_sg> „und“ <DE1_for_pl>

 <DEs1_back_pl> ::=   <DEs1_back_sg>
                    | <DEs1_back_sg> „und“ <DEs1_back_pl>

 <DEe1_back_pl> ::=   <DEe1_back_sg>
                    | <DEe1_back__sg> „und“' <DEe1_back_pl>

   <DE2_for_pl> ::=   <DE2_for_sg>
                    | <DE2_for_sg> „und“ <DE2_for_pl>

 <DEs2_back_pl> ::=   <DEs2_back_sg>
                    | <DEs2_back_sg> „und“ <DEs2_back_pl>

 <DEe2_back_pl> ::=   <DEe2_back_sg>
                    | <DEe2_back_sg> „und“ <DEe2_back_pl>

   <DE3_for_pl> ::=   <DE3_for_sg>
                    | <DE3_for_sg> „und“' <DE3_for_pl>

 <DEs3_back_pl> ::=   <DEs3_back_sg>
                    | <DEs3_back_sg> „und“ <DEs3_back_pl>

 <DEe3_back_pl> ::=   <DEe3_back_sg>
                    | <DEe3_back_sg> „und“' <DEe3_back_pl>

   <PO1_for_pl> ::=   <PO1_for_sg>
                    | <PO1_for_sg> „und“ <PO1_for_pl>

 <POe1_back_pl> ::=   <POe1_back_sg>
                    | <POe1_back_sg> „und“ <POe1_back_pl>

   <PO3_for_pl> ::=   <PO3_for_sg>
                    | <PO3_for_sg> „und” <PO3_for_pl>

 <POe3_back_pl> ::=   <POe3_back_sg>
                    | <POe3_back_sg> „und“ <POe3_back_pl>

   <PR1_for_pl> ::=   <PR1_for_sg>
```

| <PR1_for_sg> „und“ <PR1_for_pl>

<PRs1_back_pl> ::= <PRs1_back_sg>
| <PRs1_back_sg> „und“ <PRs1_back_pl>

<PR3_for_pl> ::= <PR3_for_sg>
| <PR3_for_sg> „und“ <PR3_for_pl>

<PRs3_back_pl> ::= <PRs3_back_sg>
| <PRs3_back_sg> „und“ <PRs3_back_pl>

**Ebene 5 - ausformulierte Satzteile**

<pres_cond_sg> ::= <Nomen_Wert_Paar> „ist“

<past_cond_sg> ::= <Nomen_Wert_Paar> „war“'

<DE1_for_sg> ::= „muss gleichzeitig“' <Nomen_Wert_Paar> „sein“
| „muss gleichzeitig“ <Nomen_Wert_Paar> „bleiben“

<DEs1_back_sg> ::= <Nomen_Wert_Paar> „muss gleichzeitig sein“'
| <Nomen_Wert_Paar> „muss gleichzeitig bleiben“

<DEe1_back_sg> ::= <Nomen_Wert_Paar> „muss nur sein“
| <Nomen_Wert_Paar> „muss nur bleiben“

<DE2_for_sg> ::= „muss unmittelbar“ <Nomen_Wert_Paar> „werden“
| „muss sofort“ <Nomen_Wert_Paar> „werden“
| „wird unmittelbar“' <Nomen_Wert_Paar>
| „wird sofort“' <Nomen_Wert_Paar>

<DEs2_back_sg > ::= <Nomen_Wert_Paar> „muss unmittelbar werden“
| <Nomen_Wert_Paar> „muss sofort werden“
| <Nomen_Wert_Paar> „wird unmittelbar“
| <Nomen_Wert_Paar> „wird sofort“

<DEe2_back_sg> ::= <Nomen_Wert_Paar> „muss nur unmittelbar werden“
| <Nomen_Wert_Paar> „muss nur sofort werden“
| <Nomen_Wert_Paar> „wird nur unmittelbar“
| <Nomen_Wert_Paar> „wird nur sofort“

<DE3_for_sg> ::= „muss irgendwann“ <Nomen_Wert_Paar> „werden“
| „wird irgendwann“ <Nomen_Wert_Paar>

<DEs3_back_sg> ::= <Nomen_Wert_Paar> „muss irgendwann werden“'
| <Nomen_Wert_Paar> „wird irgendwann“

<DEe3_back_sg> ::= <Nomen_Wert_Paar> „muss nur irgendwann werden“

| <Nomen_Wert_Paar> „wird nur irgendwann“

<PO1_for_sg> ::= „kann gleichzeitig“' <Nomen_Wert_Paar> „sein“
| „darf gleichzeitig“ <Nomen_Wert_Paar> „sein“

<POe1_back_sg> ::= <Nomen_Wert_Paar> „kann nur sein“
| <Nomen_Wert_Paar> „darf nur sein“

<PO3_for_sg> ::= „kann irgendwann“ <Nomen_Wert_Paar> „werden“
| „darf irgendwann“ <Nomen_Wert_Paar> „werden“

<POe3_back_sg> ::= <Nomen_Wert_Paar> „kann nur irgendwann werden“
| <Nomen_Wert_Paar> „darf nur irgendwann werden“

<PR1_for_sg> ::= „darf nicht gleichzeitig“ <Nomen_Wert_Paar> „sein“
| „darf niemals gleichzeitig“ <Nomen_Wert_Paar> „sein“

<PRs1_back_sg> ::= <Nomen_Wert_Paar> „darf nicht gleichzeitig sein“
| <Nomen_Wert_Paar> „darf niemals gleichzeitig sein“

<PR3_for_sg> ::= „darf nicht irgendwann“ <Nomen_Wert_Paar> „werden“
| „darf niemals“ <Nomen_Wert_Paar> „werden“

<PRs3_back_sg> ::= <Nomen_Wert_Paar> „darf nicht irgendwann werden“
| <Nomen_Wert_Paar> „darf niemals werden“

## A.3 Beispiele für erzeugbare Sätze

Es folgt eine Aufstellung der erzeugbaren Sätze auf Ebene 5 der formalen Syntax (ausformulierte Satzteile). Es werden Sätze mit jeweils zwei konjunktiv verknüpften Bedingungen (B1 und B2) bzw. Folgerungen (F1 und F2) dargestellt, selbstverständlich ist die Anzahl der verwendeten Bedingungen und Folgerungen beliebig.

Die Darstellung erfolgt in Metatext.

Kategorie DEs1 - einfache Forderung, Zustand

| |
|---|
| Wenn B1 ist und B2 ist, dann muss gleichzeitig F1 sein und muss gleichzeitig F2 sein. |
| Wenn B1 ist und B2 ist, dann muss gleichzeitig F1 bleiben und muss gleichzeitig F2 bleiben. |
| F1 muss gleichzeitig sein und F2 muss gleichzeitig sein, wenn B1 ist und B2 ist. |
| F1 muss gleichzeitig bleiben und F2 muss gleichzeitig bleiben, wenn B1 ist und B2 ist. |

Kategorie DEs2 - einfache Forderung, direkt

| |
|---|
| Wenn B1 ist und B2 ist, dann muss unmittelbar F1 werden und muss unmittelbar F2 werden. |
| Wenn B1 ist und B2 ist, dann muss sofort F1 werden und muss sofort F2 werden. |
| Wenn B1 ist und B2 ist, dann wird unmittelbar F1 und wird unmittelbar F2. |
| Wenn B1 ist und B2 ist, dann wird sofort F1 und wird sofort F2. |
| F1 muss unmittelbar werden und F2 muss unmittelbar werden, wenn B1 ist und B2 ist. |
| F1 muss sofort werden und F2 muss sofort werden, wenn B1 ist und B2 ist. |
| F1 wird unmittelbar und F2 wird unmittelbar, wenn B1 ist und B2 ist. |
| F1 wird sofort und F2 wird sofort, wenn B1 ist und B2 ist. |

Kategorie DEs3 - einfache Forderung, selbstbegrenzt

| |
|---|
| Solange B1 ist und B2 ist, muss irgendwann F1 werden und muss irgendwann F2 werden. |
| Solange B1 ist und B2 ist, wird irgendwann F1 und wird irgendwann F2. |
| F1 muss irgendwann werden und F2 muss irgendwann werden, solange B1 ist und B2 ist. |
| F1 wird irgendwann und F2 wird irgendwann, solange B1 ist und B2 ist. |

Kategorie DEs4 - einfache Forderung, fremdbegrenzt

| |
|---|
| Wenn irgendwann B1 war und B2 war, dann muss irgendwann F1 werden und muss irgendwann F2 werden, bevor B3 ist und B4 ist. |
| Wenn irgendwann B1 war und B2 war, dann wird irgendwann F1 und wird irgendwann F2, bevor B3 ist und B4 ist. |
| Nachdem B1 war und B2 war, muss irgendwann F1 werden und muss irgendwann F2 werden, bevor B3 ist und B4 ist. |
| Nachdem B1 war und B2 war, wird irgendwann F1 und wird irgendwann F2, bevor B3 ist und B4 ist. |

Kategorie DEs5 - einfache Forderung, unbegrenzt

| |
|---|
| Wenn irgendwann B1 war und B2 war, dann muss irgendwann F1 werden und muss irgendwann F2 werden. |
| Wenn irgendwann B1 war und B2 war, dann wird irgendwann F1 und wird irgendwann F2. |
| Nachdem B1 war und B2 war, muss irgendwann F1 werden und muss irgendwann F2 werden. |
| Nachdem B1 war und B2 war, wird irgendwann F1 und wird irgendwann F2. |
| F1 muss irgendwann werden und F2 muss irgendwann werden, wenn irgendwann B1 war und B2 war. |
| F1 wird irgendwann und F2 wird irgendwann, wenn irgendwann B1 war und B2 war. |
| F1 muss irgendwann werden und F2 muss irgendwann werden, wenn vorher B1 war und B2 war. |
| F1 wird irgendwann und F2 wird irgendwann, wenn vorher B1 war und B2 war. |

Kategorie DEe1 - erweiterte Forderung, Zustand

| |
|---|
| Nur wenn B1 ist und B2 ist, dann muss gleichzeitig F1 sein und muss gleichzeitig F2 sein. |
| Nur wenn B1 ist und B2 ist, dann muss gleichzeitig F1 bleiben und muss gleichzeitig F2 bleiben. |
| F1 muss nur sein und F2 muss nur sein, wenn gleichzeitig B1 ist und B2 ist. |
| F1 muss nur bleiben und F2 muss nur bleiben, wenn gleichzeitig B1 ist und B2 ist. |

Kategorie DEe2 - erweiterte Forderung, direkt

| |
|---|
| Nur wenn B1 ist und B2 ist, dann muss unmittelbar F1 werden und muss unmittelbar F2 werden. |
| Nur wenn B1 ist und B2 ist, dann muss sofort F1 werden und muss sofort F2 werden. |

| |
|---|
| Nur wenn B1 ist und B2 ist, dann wird unmittelbar F1 und wird unmittelbar F2. |
| Nur wenn B1 ist und B2 ist, dann wird sofort F1 und wird sofort F2. |
| F1 muss nur unmittelbar werden und F2 muss nur unmittelbar werden, wenn B1 ist und B2 ist. |
| F1 muss nur sofort werden und F2 muss nur sofort werden, wenn B1 ist und B2 ist. |
| F1 wird nur unmittelbar und F2 wird nur unmittelbar, wenn B1 ist und B2 ist. |
| F1 wird nur sofort und F2 wird nur sofort, wenn B1 ist und B2 ist. |

Kategorie DEe3 - erweiterte Forderung, selbstbegrenzt

| |
|---|
| Nur solange B1 ist und B2 ist, muss irgendwann F1 werden und muss irgendwann F2 werden. |
| Nur solange B1 ist und B2 ist, wird irgendwann F1 und wird irgendwann F2. |
| F1 muss nur irgendwann werden und F2 muss nur irgendwann werden, solange gleichzeitig B1 ist und B2 ist. |
| F1 wird nur irgendwann und F2 wird nur irgendwann, solange gleichzeitig B1 ist und B2 ist. |

Kategorie DEe4 - erweiterte Forderung, fremdbegrenzt

| |
|---|
| Nur wenn irgendwann B1 war und B2 war, dann muss irgendwann F1 werden und muss irgendwann F2 werden, bevor B3 ist und B4 ist. |
| Nur wenn irgendwann B1 war und B2 war, dann wird irgendwann F1 und wird irgendwann F2, bevor B3 ist und B4 ist. |
| Nur nachdem B1 war und B2 war, muss irgendwann F1 werden und muss irgendwann F2 werden, bevor B3 ist und B4 ist. |
| Nur nachdem B1 war und B2 war, wird irgendwann F1 und wird irgendwann F2, bevor B3 ist und B4 ist. |

Kategorie DEe5 - erweiterte Forderung, unbegrenzt

| |
|---|
| Nur wenn irgendwann B1 war und B2 war, dann muss irgendwann F1 werden und muss irgendwann F2 werden. |
| Nur wenn irgendwann B1 war und B2 war, dann wird irgendwann F1 und wird irgendwann F2. |
| Nur nachdem B1 war und B2 war, muss irgendwann F1 werden und muss irgendwann F2 werden. |
| Nur nachdem B1 war und B2 war, wird irgendwann F1 und wird irgendwann F2. |
| F1 muss nur irgendwann werden und F2 muss nur irgendwann werden, wenn irgendwann B1 war und B2 war. |
| F1 wird nur irgendwann und F2 wird nur irgendwann, wenn irgendwann B1 |

| |
|---|
| war und B2 war. |
| F1 muss nur irgendwann werden und F2 muss nur irgendwann werden, wenn vorher B1 war und B2 war. |
| F1 wird nur irgendwann und F2 wird nur irgendwann, wenn vorher B1 war und B2 war. |

Kategorie POe1 - erweiterte Möglichkeit, Zustand

| |
|---|
| Nur wenn B1 ist und B2 ist, dann kann gleichzeitig F1 sein und kann gleichzeitig F2 sein. |
| Nur wenn B1 ist und B2 ist, dann darf gleichzeitig F1 sein und darf gleichzeitig F2 sein. |
| F1 kann nur sein und F2 kann nur sein, wenn gleichzeitig B1 ist und B2 ist. |
| F1 darf nur sein und F2 darf nur sein, wenn gleichzeitig B1 ist und B2 ist. |

Kategorie POe3 - erweiterte Möglichkeit, selbstbegrenzt

| |
|---|
| Nur solange B1 ist und B2 ist, kann irgendwann F1 werden und kann irgendwann F2 werden. |
| Nur solange B1 ist und B2 ist, darf irgendwann F1 werden und darf irgendwann F2 werden. |
| F1 kann nur irgendwann werden und F2 kann nur irgendwann werden, solange gleichzeitig B1 ist und B2 ist. |
| F1 darf nur irgendwann werden und F2 darf nur irgendwann werden, solange gleichzeitig B1 ist und B2 ist. |

Kategorie POe4 - erweiterte Möglichkeit, fremdbegrenzt

| |
|---|
| Nur wenn irgendwann B1 war und B2 war, dann kann irgendwann F1 werden und kann irgendwann F2 werden, bis B3 ist und B4 ist. |
| Nur wenn irgendwann B1 war und B2 war, dann darf irgendwann F1 werden und darf irgendwann F2 werden, bis B3 ist und B4 ist. |
| Nur nachdem B1 war und B2 war, dann kann irgendwann F1 werden und kann irgendwann F2 werden, bis B3 ist und B4 ist. |
| Nur nachdem B1 war und B2 war, dann darf irgendwann F1 werden und darf irgendwann F2 werden, bis B3 ist und B4 ist. |

Kategorie POe5 - erweiterte Möglichkeit, unbegrenzt

| |
|---|
| Nur wenn irgendwann B1 war und B2 war, dann kann irgendwann F1 werden und kann irgendwann F2 werden. |
| Nur wenn irgendwann B1 war und B2 war, dann darf irgendwann F1 werden und darf irgendwann F2 werden. |

| |
|---|
| Nur nachdem B1 war und B2 war, kann irgendwann F1 werden und kann irgendwann F2 werden. |
| Nur nachdem B1 war und B2 war, darf irgendwann F1 werden und darf irgendwann F2 werden. |
| F1 kann nur irgendwann werden und F2 kann nur irgendwann werden, wenn irgendwann B1 war und B2 war. |
| F1 darf nur irgendwann werden und F2 darf nur irgendwann werden, wenn irgendwann B1 war und B2 war. |
| F1 kann nur irgendwann werden und F2 kann nur irgendwann werden, wenn vorher B1 war und B2 war. |
| F1 darf nur irgendwann werden und F2 darf nur irgendwann werden, wenn vorher B1 war und B2 war. |

Kategorie PRs1 - einfaches Verbot, Zustand

| |
|---|
| Wenn B1 ist und B2 ist, dann darf nicht gleichzeitig F1 sein und darf nicht gleichzeitig F2 sein. |
| Wenn B1 ist und B2 ist, dann darf niemals gleichzeitig F1 sein und darf niemals gleichzeitig F2 sein. |
| F1 darf nicht gleichzeitig sein und F2 darf nicht gleichzeitig sein, wenn B1 ist und B2 ist. |
| F1 darf niemals gleichzeitig sein und F2 darf niemals gleichzeitig sein, wenn B1 ist und B2 ist. |

Kategorie PRs3 - einfaches Verbot, selbstbegrenzt

| |
|---|
| Solange B1 ist und B2 ist, darf nicht irgendwann F1 werden und darf nicht irgendwann F2 werden. |
| Solange B1 ist und B2 ist, darf niemals F1 werden und darf niemals F2 werden. |
| F1 darf nicht irgendwann werden und F2 darf nicht irgendwann werden, solange B1 ist und B2 ist. |
| F1 darf niemals werden und F2 darf niemals werden, solange B1 ist und B2 ist. |

Kategorie PRs4 - einfaches Verbot, fremdbegrenzt

| |
|---|
| Wenn irgendwann B1 war und B2 war, dann darf nicht irgendwann F1 werden und darf nicht irgendwann F2 werden, bis B3 ist und B4 ist. |
| Wenn irgendwann B1 war und B2 war, dann darf niemals F1 werden und darf niemals F2 werden, bis B3 ist und B4 ist. |
| Nachdem B1 war und B2 war, darf nicht irgendwann F1 werden und darf nicht irgendwann F2 werden, bis B3 ist und B4 ist. |

| |
|---|
| Nachdem B1 war und B2 war, darf niemals F1 werden und darf niemals F2 werden, bis B3 ist und B4 ist. |

Kategorie PRs5 - einfaches Verbot, unbegrenzt

| |
|---|
| Wenn irgendwann B1 war und B2 war, dann darf nicht irgendwann F1 werden und darf nicht irgendwann F2 werden. |
| Wenn irgendwann B1 war und B2 war, dann darf niemals F1 werden und darf niemals F2 werden. |
| Nachdem B1 war und B2 war, darf nicht irgendwann F1 werden und darf nicht irgendwann F2 werden. |
| Nachdem B1 war und B2 war, darf niemals F1 werden und darf niemals F2 werden. |
| F1 darf nicht irgendwann werden und F2 darf nicht irgendwann werden, wenn irgendwann B1 war und B2 war. |
| F1 darf niemals werden und F2 darf niemals werden, wenn irgendwann B1 war und B2 war. |
| F1 darf nicht irgendwann werden und F2 darf nicht irgendwann werden, wenn vorher B1 war und B2 war. |
| F1 darf niemals werden und F2 darf niemals werden, wenn vorher B1 war und B2 war. |

### A.4 Struktur der Datei zur Definition und Zuordnung der Nomen und Werte

```
<Zuordnung> ::= „%%“ <Nr> „-“ <Datentyp> „-“
                <Variablentyp> „ " “ <SPS-Variable>
                „ " : " “ <Nomen> „"“ <Wertzuordnung>

<Nr> ::= „1“, „2“, „3“, ...

<Datentyp> ::= „BOOL” | „INT” | „REAL”

<Variablentyp> ::=   „VAR”
                   | „VAR_ACCESS”'
                   | „VAR_EXTERNAL”
                   | „VAR_GLOBAL”
                   | „VAR_IN_OUT”'
                   | „VAR_INPUT”
                   | „VAR_OUTPUT”

<SPS_Variable> ::= <name>

<Nomen> ::= <name>

<Wertzuordnung> ::=   <Wertzuordnung_BOOL>
                    | <Wertzuordnung_INT>

<Wertzuordnung_BOOL> ::= „TRUE_I : "“ <Wert> „"“
                         „TRUE_O : "“' <Wert> „"“
                         „FALSE_I : "“ <Wert> „"“
                         „FALSE_O : "“' <Wert> „"“

<Wertzuordnung_INT> ::= <Wert1> „"_I : "“ <Wert> „"“
                        <Wert1> „"_O : "“ <Wert> „"“
                        <Wert2> „"_I : "“ <Wert> „"“
                        <Wert2> „"_O : "“ <Wert> „"“
                        ...

<Wert> ::= <zahl>
```

```
<name>    ::=   <zeichen>
              | <zeichen> <name>

<zahl>    ::=   <ziffer>
              | <ziffer> <zahl>

<zeichen> ::=  „a“, „b“,…, „z“, „A“, „B“,…, „Z“, „_“

<ziffer>  ::=  „0“, „1“,…., „9“
```

### A.5 Formale Definition des PreLexers

#### Ebene a - grundlegende Satzkonstruktionen

<statement> ::= „Wenn“ <cond_pl> „ , dann“ <for_concl_pl> „ .“
| „Wenn irgendwann“ <cond_pl> „ , dann“ <for_concl_pl> „ .“
| „Wenn irgendwann“ <cond_pl> „ , dann“ <for_concl_pl> „ , bevor“ <cond_pl> „ .“
| „Wenn irgendwann“ <cond_pl> „ , dann“ <for_concl_pl> „ , bis“ <cond_pl> „ .“
| „Nur wenn“ <cond_pl> „ , dann“ <for_concl_pl> „ .“
| „Nur wenn irgendwann“ <cond_pl> „ , dann“ <for_concl_pl> „ .“
| „Nur wenn irgendwann“ <cond_pl> „ , dann“ <for_concl_pl> „ , bevor“ <cond_pl> „ .“
| „Nur wenn irgendwann“ <cond_pl> „ , dann“ <for_concl_pl> „ , bis“ <cond_pl> „ .“
| „Nachdem“ <cond_pl> „ ,“ <for_concl_pl> „ .“
| „Nachdem“ <cond_pl> „ ,“ <for_concl_pl> „ , bevor“ <cond_pl> „ .“
| „Nachdem“ <cond_pl> „ ,“ <for_concl_pl> „ , bis“ <cond_pl> „ .“
| „Nur nachdem“ <cond_pl> „ ,“ <for_concl_pl> „ .“
| „Nur nachdem“ <cond_pl> „ ,“ <for_concl_pl> „ , bevor“ <cond_pl> „ .“
| „Nur nachdem“ <cond_pl> „ , dann“ <for_concl_pl> „ , bis“ <cond_pl> „ .“
| „Solange“ <cond_pl> „ ,“ <for_concl_pl> „ .“
| „Nur solange“' <cond_pl> „ ,“ <for_concl_pl> „ .“
| <back_concl_pl> „ , wenn“ <cond_pl> „ .“
| <back_concl_pl> „ , wenn gleichzeitig“ <cond_pl> „ .“
| <back_concl_pl> „ , wenn irgendwann“ <cond_pl> „ .“
| <back_concl_pl> „ , wenn vorher“ <cond_pl> „ .“
| <back_concl_pl> „ , solange“ <cond_pl> „ .“
| <back_concl_pl> „ , solange gleichzeitig“ <cond_pl> „ .“

### Ebene b

```
      <cond_pl> ::=   <cond_sg_start>
                    | <cond_sg_start> „und“ <cond_pl_follow>

  <for_concl_pl> ::=  <for_concl_sg>
                    | <for_concl_sg> „und“ <for_concl_pl>

 <back_concl_pl> ::=  <back_concl_sg>
                    | <back_concl_sg> „und“ <back_concl_pl>
```

### Ebene c

```
 <cond_sg_start> ::=  <Nomen> <Wert> „ist“
                    | <Nomen> <Wert> „war“

<cond_pl_follow> ::=  <cond_sg_start>
                    | <cond_sg_follow>
                    | <cond_sg_start> „und“ <cond_pl_follow>
                    | <cond_sg_follow> „und“ <cond_pl_follow>

<cond_sg_follow> ::=  <Nomen> „ist“ <Wert>
                    | <Nomen> „war“ <Wert>


  <for_concl_sg> ::=  <Nomen> „muss gleichzeitig“ <Wert> „sein“
                    | „muss“ <Nomen> „gleichzeitig“ <Wert> „sein“
                    | „muss gleichzeitig“' <Nomen> <Wert> „sein“
                    | <Nomen> „muss gleichzeitig“ <Wert> „bleiben“
                    | „muss“ <Nomen> „gleichzeitig“ <Wert> „bleiben“
                    | „muss gleichzeitig“ <Nomen> <Wert> „bleiben“
                    | <Nomen> „muss unmittelbar“' <Wert> „werden“
                    | „muss“ <Nomen> „unmittelbar“ <Wert> „werden“'
                    | „muss unmittelbar“ <Nomen> <Wert> „werden“'
                    | <Nomen> „muss sofort“ <Wert> „werden“'
                    | „muss“ <Nomen> „sofort“ <Wert> „werden“
                    | „muss sofort“ <Nomen> <Wert> „werden“
                    | <Nomen> „wird unmittelbar“ <Wert>
                    | „wird“ <Nomen> „unmittelbar“' <Wert>
                    | „wird unmittelbar“ <Nomen> <Wert>
                    | <Nomen> „wird sofort“ <Wert>
                    | „wird“ <Nomen> „sofort“ <Wert>
                    | „wird sofort“ <Nomen> <Wert>
                    | <Nomen> „muss irgendwann“' <Wert> „werden“'
                    | „muss“ <Nomen> „irgendwann“ <Wert> „werden“
                    | „muss irgendwann“ <Nomen> <Wert> „werden“'
                    | <Nomen> „wird irgendwann“ <Wert>
```

| „wird“ <Nomen> „irgendwann“ <Wert>
| „wird irgendwann“' <Nomen> <Wert>
| <Nomen> „kann gleichzeitig“ <Wert> „sein“
| „kann“ <Nomen> „gleichzeitig“ <Wert> „sein“
| „kann gleichzeitig“ <Nomen> <Wert> „sein“
| <Nomen> „darf gleichzeitig“ <Wert> „sein“
| „darf“ <Nomen> „gleichzeitig“ <Wert> „sein“
| „darf gleichzeitig“ <Nomen> <Wert> „sein“
| <Nomen> „kann irgendwann“' <Wert> „werden“
| „kann“ <Nomen> „irgendwann“ <Wert> „werden“'
| „kann irgendwann“' <Nomen> <Wert> „werden“
| <Nomen> „darf irgendwann“ <Wert> „werden“
| „darf“ <Nomen> „irgendwann“ <Wert> „werden“
| „darf irgendwann“ <Nomen> <Wert> „werden“
| <Nomen> „darf nicht gleichzeitig“' <Wert> „sein“
| „darf“ <Nomen> „nicht gleichzeitig“' <Wert> „sein“
| „darf nicht gleichzeitig“ <Nomen> <Wert> „sein“
| <Nomen> „darf niemals gleichzeitig“ <Wert> „sein“
| „darf“ <Nomen> „niemals gleichzeitig“ <Wert> „sein“
| „darf niemals gleichzeitig“ <Nomen> <Wert> „sein“
| <Nomen> „darf nicht irgendwann“ <Wert> „werden“
| „darf“ <Nomen> „nicht irgendwann“ <Wert> „werden“
| „darf nicht irgendwann“ <Nomen> <Wert> „werden“
| <Nomen> „darf niemals“ <Wert> „werden“
| „darf“' <Nomen> „niemals“ <Wert> „werden“
| „darf niemals“ <Nomen> <Wert> „werden“

<back_concl_sg> ::= <Nomen> „muss gleichzeitig“ <Wert> „sein“
| <Nomen> „muss gleichzeitig“ <Wert> „bleiben“'
| <Nomen> „muss nur“' <Wert> „sein“
| <Nomen> „muss nur“ <Wert> „bleiben“'
| <Nomen> „muss unmittelbar“' <Wert> „werden“
| <Nomen> „muss sofort“ <Wert> „werden“
| <Nomen> „wird unmittelbar“ <Wert>
| <Nomen> „wird sofort“ <Wert>
| <Nomen> „muss nur unmittelbar“' <Wert> „werden“'
| <Nomen> „muss nur sofort“ <Wert> „werden“
| <Nomen> „wird nur unmittelbar“ <Wert>
| <Nomen> „wird nur sofort“ <Wert>
| <Nomen> „muss irgendwann“ <Wert> „werden“
| <Nomen> „wird irgendwann“ <Wert>
| <Nomen> „muss nur irgendwann“ <Wert> „werden“
| <Nomen> „wird nur irgendwann“ <Wert>
| <Nomen> „kann gleichzeitig“ <Wert> „sein“

| <Nomen> „darf gleichzeitig“ <Wert> „sein“
| <Nomen> „kann nur“ <Wert> „sein“
| <Nomen> „darf nur“ <Wert> „sein“
| <Nomen> „kann irgendwann“ <Wert> „werden“
| <Nomen> „darf irgendwann“' <Wert> „werden“
| <Nomen> „kann nur irgendwann“' <Wert> „werden“
| <Nomen> „darf nur irgendwann“ <Wert> „werden“'
| <Nomen> „darf nicht gleichzeitig“' <Wert> „sein“
| <Nomen> „darf niemals gleichzeitig“' <Wert> „sein“
| <Nomen> „darf nur nicht gleichzeitig“ <Wert> „sein“
| <Nomen> „darf nicht irgendwann“ <Wert> „werden“
| <Nomen> „darf niemals“ <Wert> „werden“
| <Nomen> „darf nur nicht irgendwann“ <Wert> „werden“
| <Nomen> „darf nur niemals“ <Wert> „werden“

**Umsetzung (Nomen und Werte werden umgestellt und zusammengefasst)**

| | | |
|---|---|---|
| <Nomen> <Wert> „ist“'<br><Nomen> „ist“ <Wert> | ⇒ | <Nomen_Wert_Paar> „ist“ |
| <Nomen> <Wert> „war“<br><Nomen> „war“ <Wert> | ⇒ | <Nomen_Wert_Paar> „war“ |
| <Nomen> „muss gleichzeitig“' <Wert> „sein“<br>„muss“ <Nomen> „gleichzeitig“ <Wert> „sein“<br>„muss gleichzeitig“ <Nomen> <Wert> „sein“ | ⇒ | „muss gleichzeitig“ <Nomen_Wert_Paar> „sein“ |
| <Nomen> „muss gleichzeitig“ <Wert> „bleiben“<br>„muss“ <Nomen> „gleichzeitig“' <Wert> „bleiben“<br>„muss gleichzeitig“ <Nomen> <Wert> „bleiben“ | ⇒ | „muss gleichzeitig“ <Nomen_Wert_Paar> „bleiben“ |
| <Nomen> „muss gleichzeitig“ <Wert> „sein“ | ⇒ | <Nomen_Wert_Paar> „muss gleichzeitig sein“ |
| <Nomen> „muss gleichzeitig“ <Wert> „bleiben“ | ⇒ | <Nomen_Wert_Paar> „muss gleichzeitig bleiben“ |
| <Nomen> „muss nur“' <Wert> „sein“ | ⇒ | <Nomen_Wert_Paar> „muss nur sein“ |
| <Nomen> „muss nur“ <Wert> | ⇒ | <Nomen_Wert_Paar> „muss nur bleiben“ |

| | | |
|---|---|---|
| „bleiben“ | | |
| <Nomen> „muss unmittelbar“ <Wert> „werden“<br>„muss“ <Nomen> „unmittelbar“ <Wert> „werden“<br>„muss unmittelbar“ <Nomen> <Wert> „werden“ | ⇒ | „muss unmittelbar“ <Nomen_Wert_Paar> „werden“ |
| <Nomen> „muss sofort“ <Wert> „werden“<br>„muss“ <Nomen> „sofort“ <Wert> „werden“<br>„muss sofort“ <Nomen> <Wert> „werden“ | ⇒ | „muss sofort“ <Nomen_Wert_Paar> „werden“ |
| <Nomen> „wird unmittelbar“ <Wert><br>„wird“ <Nomen> „unmittelbar“ <Wert><br>„wird unmittelbar“ <Nomen> <Wert> | ⇒ | „wird unmittelbar“ <Nomen_Wert_Paar> |
| <Nomen> „wird sofort“ <Wert><br>„wird“ <Nomen> „sofort“ <Wert><br>„wird sofort“ <Nomen> <Wert> | ⇒ | „wird sofort“ <Nomen_Wert_Paar> |
| <Nomen> „muss unmittelbar“ <Wert> „werden“ | ⇒ | <Nomen_Wert_Paar> „muss unmittelbar werden“ |
| <Nomen> „muss sofort“ <Wert> „werden“ | ⇒ | <Nomen_Wert_Paar> „muss sofort werden“ |
| <Nomen> „wird unmittelbar“ <Wert> | ⇒ | <Nomen_Wert_Paar> „wird unmittelbar“ |
| <Nomen> „wird sofort“ <Wert> | ⇒ | <Nomen_Wert_Paar> „wird sofort“ |
| <Nomen> „muss nur unmittelbar“ <Wert> „werden“ | ⇒ | <Nomen_Wert_Paar> „muss nur unmittelbar werden“ |
| <Nomen> „muss nur sofort“ <Wert> „werden“ | ⇒ | <Nomen_Wert_Paar> „muss nur sofort werden“ |
| <Nomen> „wird nur unmittelbar“ <Wert> | ⇒ | <Nomen_Wert_Paar> „wird nur unmittelbar“ |
| <Nomen> „wird nur sofort“ <Wert> | ⇒ | <Nomen_Wert_Paar> „wird nur sofort“ |
| <Nomen> „muss irgendwann“ <Wert> „werden“<br>„muss“ <Nomen> „irgendwann“ <Wert> „werden“ | ⇒ | „muss irgendwann“ <Nomen_Wert_Paar> „werden“ |

| | | |
|---|---|---|
| „muss irgendwann“ <Nomen> <Wert> „werden“ | | |
| <Nomen> „wird irgendwann“ <Wert><br>„wird“ <Nomen> „irgendwann“' <Wert><br>„wird irgendwann“ <Nomen> <Wert> | ⇒ | „wird irgendwann“ <Nomen_Wert_Paar> |
| <Nomen> „muss irgendwann“ <Wert> „werden“ | ⇒ | <Nomen_Wert_Paar> „muss irgendwann werden“ |
| <Nomen> „wird irgendwann“ <Wert> | ⇒ | <Nomen_Wert_Paar> „wird irgendwann“ |
| <Nomen> „muss nur irgendwann“ <Wert> „werden“ | ⇒ | <Nomen_Wert_Paar> „muss nur irgendwann werden“ |
| <Nomen> „wird nur irgendwann“ <Wert> | ⇒ | <Nomen_Wert_Paar> „wird nur irgendwann“ |
| <Nomen> „kann nur“ <Wert> „sein“ | ⇒ | <Nomen_Wert_Paar> „kann nur sein“ |
| <Nomen> „darf nur“ <Wert> „sein“ | ⇒ | <Nomen_Wert_Paar> „darf nur sein“ |
| <Nomen> „kann nur irgendwann“ <Wert> „werden“ | ⇒ | <Nomen_Wert_Paar> „kann nur irgendwann werden“ |
| <Nomen> „darf nur irgendwann“ <Wert> „werden“ | ⇒ | <Nomen_Wert_Paar> „darf nur irgendwann werden“ |
| <Nomen> „darf nicht gleichzeitig“ <Wert> „sein“<br>„darf“ <Nomen> „nicht gleichzeitig“ <Wert> „sein“<br>„darf nicht gleichzeitig“ <Nomen> <Wert> „sein“ | ⇒ | „darf nicht gleichzeitig“ <Nomen_Wert_Paar> „sein“ |
| <Nomen> „darf niemals gleichzeitig“ <Wert> „sein“<br>„darf“ <Nomen> „niemals gleichzeitig“ <Wert> „sein“<br>„darf niemals gleichzeitig“ <Nomen> <Wert> „sein“ | ⇒ | „darf niemals gleichzeitig“ <Nomen_Wert_Paar> „sein“ |
| <Nomen> „darf nicht gleichzeitig“ <Wert> „sein“ | ⇒ | <Nomen_Wert_Paar> „darf nicht gleichzeitig sein“ |
| <Nomen> „darf niemals gleichzeitig“' <Wert> „sein“ | ⇒ | <Nomen_Wert_Paar> „darf niemals gleichzeitig sein“ |
| <Nomen> „darf nicht irgendwann“ <Wert> „werden“ | ⇒ | „darf nicht irgendwann“ <Nomen_Wert_Paar> „werden“ |

| | | |
|---|---|---|
| „darf“ <Nomen> „nicht irgendwann“ <Wert> „werden“ | | |
| „darf nicht irgendwann“ <Nomen> <Wert> „werden“ | | |
| <Nomen> „darf niemals“ <Wert> „werden“ | | |
| „darf“ <Nomen> „niemals“ <Wert> „werden“ | ⇒ | „darf niemals“ <Nomen_Wert_Paar> „werden“ |
| „darf niemals“ <Nomen> <Wert> „werden“ | | |
| <Nomen> „darf nicht irgendwann“ <Wert> „werden“' | ⇒ | <Nomen_Wert_Paar> „darf nicht irgendwann werden“' |
| <Nomen> „darf niemals“ <Wert> „werden“ | ⇒ | <Nomen_Wert_Paar> „darf niemals werden“ |

### A.6 Erzeugbare Beispielsätze auf verbaler Ebene

An dieser Stelle soll noch einmal kurz der Sprachumfang der Sicherheitsfachsprache gezeigt werden. Es werden bewusst nur Beispielsätze der Kategorie DEs1 (einfache Forderung, Zustand) dargestellt, um den Rahmen dieser Darstellung nicht zu sprengen. Der Leser wird sich sicher vorstellen können, wie viele Möglichkeiten der Bildung korrekter Sätze es gibt.

In Anhang A.3 wurden in der genannten Kategorie vier Mustersätze gezeigt. Durch Aufteilung des *Nomen-Wert-Paares* (dargestellt durch B1, B2, F1, F2) in einzelne *Nomen* und *Werte* (hier unspezifisch dargestellt als „x1“, „x2“, „y1“ sowie „y2“ und „*True*“) und zusätzliche Umstellung der Wortreihenfolge lassen sich die nachfolgend gezeigten verbalen Anforderungssätze erzeugen.

Wenn B1 ist und B2 ist, dann muss gleichzeitig F1 sein und muss gleichzeitig F2 sein.

*Wenn „x1“ „True“ ist und „x2“ ist „True“, dann muss gleichzeitig „y1“ „True“ sein und muss gleichzeitig „y2“ „True“ sein.*

*Wenn „x1“ „True“ ist und „x2“ „True“ ist, dann muss gleichzeitig „y1“ „True“ sein und muss gleichzeitig „y2“ „True“ sein.*

*Wenn „x1“ „True“ ist und „x2“ ist „True“, dann muss „y1“ gleichzeitig „True“ sein und muss „y2“ gleichzeitig „True“ sein.*

*Wenn „x1“ „True“ ist und „x2“ ist „True“, dann muss „y1“ gleichzeitig „True“ sein und „y2“ muss gleichzeitig „True“ sein.*

Wenn B1 ist und B2 ist, dann muss gleichzeitig F1 bleiben und muss gleichzeitig F2 bleiben.

*Wenn „x1“ „True“ ist und „x2“ ist „True“, dann muss gleichzeitig „y1“ „True“ bleiben und muss gleichzeitig „y2“ „True“ bleiben.*

*Wenn „x1“ „True“ ist und „x2“ „True“ ist, dann muss gleichzeitig „y1“ „True“ bleiben und muss gleichzeitig „y2“ „True“ bleiben.*

*Wenn „x1“ „True“ ist und „x2“ ist „True“, dann muss „y1“ gleichzeitig „True“ bleiben und muss „y2“ gleichzeitig „True“ bleiben.*

*Wenn „x1“ „True“ ist und „x2“ ist „True“, dann muss „y1“ gleichzeitig „True“ bleiben und „y2“ muss gleichzeitig „True“ bleiben.*

F1 muss gleichzeitig sein und F2 muss gleichzeitig sein, wenn B1 ist und B2 ist.

*„y1“ muss gleichzeitig „True“ sein und „y2“ muss gleichzeitig „True“ sein, wenn „x1“ „True“ ist und „x2“ „True“ ist.*

*„y1“ muss gleichzeitig „True“ sein und „y2“ muss gleichzeitig „True“ sein, wenn*

*„x1“ „True“ ist und „x2“ ist „True“.*

F1 muss gleichzeitig bleiben und F2 muss gleichzeitig bleiben, wenn B1 ist und B2 ist.

*„y1“ muss gleichzeitig „True“ bleiben und „y2“ muss gleichzeitig „True“ bleiben, wenn „x1“ „True“ ist und „x2“ „True“ ist.*

*„y1“ muss gleichzeitig „True“ bleiben und „y2“ muss gleichzeitig „True“ bleiben, wenn „x1“ „True“ ist und „x2“ ist „True“.*

## A.7 Fallstudie „Erweiterte Produktionszelle“

### *A.7.1 Spezifikation der Geräteebene 1 - Auftragsverwaltung*

*Gegebene Daten*

nur Idee Produktionsauftrag

*Informelle Spezifikation:*

es sollen Produktionsaufträge bearbeitet werden

*Definition neuer Daten mit dem SFS-Editor*

Datenebene 1 Produktionsauftrag

| **Technische Bezeichnung** | **Variable** | **Datentyp** | **Kommentar** |
|---|---|---|---|
| Auftragsnummer | `order_number` | Integer | laufende Nummer des Auftrags |
| Bearbeitungsart | `order_work` | Integer | wie sollen diese Teile bearbeitet werden |
| Teileanzahl | `order_quantity` | Integer | wie viele Teile sollen bearbeitet werden |
| Auftragsfortschritt | `order_status` | Integer | Stand der Auftragsbearbeitung |

*Tabelle 9 - Datenebene 1: Datensatz „Produktionsauftrag“*

```
%%1 -INT -VAR_GLOBAL
"order_number" : "die laufende Nummer des
Produktionssauftrages"
1_I : "1"
2_I : "2"
1_O : "1"
2_O : "2"

%%2 -INT -VAR_GLOBAL
"order_work" : "die Bearbeitungsart"
1_I : "Pressen"
2_I : "Bohren"
3_I : "Fräsen"

%%3 -INT -VAR_GLOBAL
"order_quantity" : "Teileanzahl"
1_I : "1"
5_I : "5"
10_I : "10"
```

```
20_I : "20"

%%4 -INT -VAR_GLOBAL
"order_status" : "der aktuelle Stand der Auftragsbearbeitung"
1_I : "Neu"
2_I : "Rohteile liefern"
3_I : "Bearbeiten"
4_I : "Fertigteile wegbringen"
5_I : "Fertig"
1_O : "auf Neu gesetzt"
2_O : "auf Rohteile holen gesetzt"
3_O : "auf Bearbeiten gesetzt"
4_O : "auf Fertigteile wegbringen gesetzt"
5_O : "auf Fertig gesetzt"
```

*Formale Spezifikation mit dem SFS-Editor*

Auftragsfortschritt

*/* >>>Satz 1<<< (einfache Forderung - unbegrenzt) */*

*Wenn irgendwann "der aktuelle Stand der Auftragsbearbeitung" "Neu" war , dann muss "der aktuelle Stand der Auftragsbearbeitung" irgendwann "auf Fertig gesetzt" werden .*

```
AG ( (rdy_in & order_status=1) -> AF (rdy_plc &
order_status=5) )
```

Darüber hinaus sind weitere Anforderungen formulierbar:

*/* >>>Satz 2<<< (einfache Forderung - unbegrenzt) */*

*Wenn irgendwann "der aktuelle Stand der Auftragsbearbeitung" "Neu" war , dann muss "der aktuelle Stand der Auftragsbearbeitung" irgendwann "auf Bearbeiten gesetzt" werden .*

```
AG ( (rdy_in & order_status=1) -> AF (rdy_plc &
order_status=3) )
```

*/* >>>Satz 3<<< (einfache Forderung - unbegrenzt) */*

*Wenn irgendwann "der aktuelle Stand der Auftragsbearbeitung" "Bearbeiten" war , dann muss "der aktuelle Stand der Auftragsbearbeitung" irgendwann "auf Fertig gesetzt" werden .*

```
AG ( (rdy_in & order_status=3) -> AF (rdy_plc &
order_status=5) )
```

*Informelle Spezifikation:*

die untergeordneten Komponenten müssen koordiniert werden

*Definition neuer Daten mit dem SFS-Editor*

Datenebene 2 - Transportauftrag

| **Technische Bezeichnung** | **Variable** | **Daten-typ** | **Kommentar** |
|---|---|---|---|
| Auftragsart | `tr_kind` | Integer | welche Art von Transport-auftrag soll ausgeführt werden |
| Bearbeitungsort | `tr_place` | Integer | welche Produktionszelle soll angefahren werden |
| Teileanzahl | `tr_quantity` | Integer | wie viele Teile sollen transportiert werden |

*Tabelle 10 - Datenebene 2: Datensatz „Transportauftrag“*

```
%%5 -INT -VAR_GLOBAL
"tr_kind" : "die Transportart"
0_I : "0 kein Transport"
1_I : "1 Rohteile holen"
2_I : "2 Fertigteile wegbringen"
0_O : "auf 0 kein Transport gesetzt"
1_O : "auf 1 Rohteile holen gesetzt"
2_O : "auf 2 Fertigteile wegbringen gesetzt"

%%6 -INT -VAR_GLOBAL
"tr_place" : "die anzufahrende Produktionszelle"
1_I : "1"
2_I : "2"
1_O : "1"
2_O : "2"

%%7 -INT -VAR_GLOBAL
"tr_quantity" : "die Anzahl der zu transportierenden Teile"
1_I : "1"
5_I : "5"
10_I : "10"
20_I : "20"
1_O : "auf 1 gesetzt"
5_O : "auf 5 gesetzt"
10_O : "auf 10 gesetzt"
20_O : "auf 20 gesetzt"
```

Datenebene 2 - Bereitschaftsmeldungen der Produktionszellen

| **Technische Bezeichnung** | **Variable** | **Datentyp** | **Kommentar** |
|---|---|---|---|
| Bereitschaftsmeldung Zelle 1 | `cell_1_ready` | Bool | Zelle 1 ist für neuen Auftrag bereit |
| Bereitschaftsmeldung Zelle 2 | `cell_2_ready` | Bool | Zelle 2 ist für neuen Auftrag bereit |
| ... | ... | ... | ... |
| Bereitschaftsmeldung Zelle n | `cell_n_ready` | Bool | Zelle n ist für neuen Auftrag bereit |

*Tabelle 11 - Datenebene 2: Bereitschaftsmeldungen der Produktionszellen*

```
%%8 -BOOL -VAR_GLOBAL
"cell_1_rdy" : "die Bereitschaftsmeldung der Produktionszelle
1"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%9 -BOOL -VAR_GLOBAL
"cell_2_rdy" : "die Bereitschaftsmeldung der Produktionszelle
2"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"
```

Datenebene 2 - Fertigmeldungen der Produktionszellen

| **Technische Bezeichnung** | **Variable** | **Datentyp** | **Kommentar** |
|---|---|---|---|
| Fertigmeldung Zelle 1 | `cell_1_finished` | Bool | Zelle 1 ist mit Auftrag fertig |
| Fertigmeldung Zelle 2 | `cell_2_finished` | Bool | Zelle 2 ist mit Auftrag fertig |
| ... | ... | ... | ... |
| Fertigmeldung Zelle n | `cell_n_finished` | Bool | Zelle n ist mit Auftrag fertig |

*Tabelle 12 - Datenebene 2: Fertigmeldungen der Produktionszellen*

```
%%10 -BOOL -VAR_GLOBAL
"cell_1_finished" : "die Fertigmeldung der Produktionszelle 1"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%11 -BOOL -VAR_GLOBAL
"cell_2_finished" : "die Fertigmeldung der Produktionszelle 2"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"
```

Datenebene 2 - Bereitschafts- und Fertigmeldung des Transportsystems

| **Technische Bezeichnung** | **Variable** | **Datentyp** | **Kommentar** |
|---|---|---|---|
| Bereitschaftsmeldung Transportsystem | `tr_sys_ready` | Bool | Transportsystem ist für neuen Auftrag bereit |
| Fertigmeldung Transportsystem | `tr_sys_finished` | Bool | Transportsystem hat Auftrag ausgeführt |

*Tabelle 13 - Datenebene 2: Bereitschafts- und Fertigmeldung des Transportsystems*

```
%%12 -BOOL -VAR_GLOBAL
"tr_sys_ready" : "die Bereitschaftsmeldung des
Transportsystems"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%13 -BOOL -VAR_GLOBAL
"tr_sys_finished" : "die Fertigmeldung des Transportsystems"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"
```

*Formale Spezifikation mit dem SFS-Editor*

*/* >>>Satz 4<<< (einfache Forderung - direkt) */*

*Wenn "der aktuelle Stand der Auftragsbearbeitung" "Neu" ist und "die Bearbeitungsart" ist "Pressen" und "die Bereitschaftsmeldung der Produktionszelle 1" ist "gesetzt" und "die Bereitschaftsmeldung des Transportsystems" ist "gesetzt" , dann muss "der aktuelle Stand der Auftragsbearbeitung" unmittelbar "auf Rohteile holen gesetzt" werden und "die Transportart" muss unmittelbar "auf 1 (Rohteile holen) gesetzt" werden und "die anzufahrende Produktionszelle" muss unmittelbar "1"*

*werden und "die Bereitschaftsmeldung der Produktionszelle 1" muss unmittelbar "zurückgesetzt" werden und "die Bereitschaftsmeldung des Transportsystems" muss unmittelbar "zurückgesetzt" werden .*

```
AG ( (rdy_in & order_status=1 & order_work=1 &
cell_1_ready & tr_sys_ready) -> A[!rdy_plc U (rdy_plc &
order_status=2 & tr_kind=1 & tr_place=1 & !cell_1_ready &
!tr_sys_ready) ] )
```

*/* >>>Satz 5<<< (einfache Forderung - direkt) */*

Wenn "der aktuelle Stand der Auftragsbearbeitung" "Bearbeiten" ist und "die Fertigmeldung der Produktionszelle 1" ist "gesetzt" und "die Bereitschaftsmeldung des Transportsystems" "gesetzt" ist , dann muss "der aktuelle Stand der Auftragsbearbeitung" unmittelbar "auf Fertigteile wegbringen gesetzt" werden und "die Transportart" muss unmittelbar "auf 2 (Fertigteile wegbringen) gesetzt" werden und "die anzufahrende Produktionszelle" muss unmittelbar "1" werden und "die Fertigmeldung der Produktionszelle 1" muss unmittelbar "zurückgesetzt" werden und "die Bereitschaftsmeldung des Transportsystems" muss unmittelbar "zurückgesetzt" werden .

```
AG ( (rdy_in & order_status=3 & cell_1_finished &
tr_sys_ready) -> A[!rdy_plc U (rdy_plc & order_status=4 &
tr_kind=2 & tr_place=1 & !cell_1_finished & !tr_sys_ready)
] )
```

### *A.7.2 Geräteebene 2 – Zentrale Steuerung der Produktionszelle*

#### *A.7.2.1 Kommunikation mit MMI und übergeordneter Ebene*

*Gegebene Daten:*

Für die zentrale Steuerung der Produktionszelle können die Bereitschafts- und Fertigmeldung zur Kommunikation mit der Auftragsverwaltung als bekannt vorausgesetzt werden.

*Definition neuer Daten mit dem SFS-Editor*

An dieser Stelle erfolgt die Definition der Sensor- und Aktorsignale, die direkt an die zentrale Steuerung der Produktionszelle angeschlossen sind

Sensoren des MMI / Bedienelemente

| **Technische Bezeichnung** | **Variable** | **Datentyp** | **Kommentar** |
|---|---|---|---|
| Sensor „Eingangspuffer belegt“ | `Piece_at_source` | Bool | TRUE - Teil vorhanden |
| Sensor „Ausgangspuffer belegt“ | `Piece_at_drain` | Bool | TRUE - Teil vorhanden |
| Not-Aus-Schalter (rastend) | `Em_stopp` | Bool | TRUE - Not-Aus nicht aktiv |
| Re-Start-Taster (nichtrastend) | `Restart` | Bool | TRUE - Restart gedrückt |

*Tabelle 14 - Datenebene 3: Sensoren des HMI / Bedienelemente*

```
%%14 -BOOL -VAR_INPUT
"Piece_at_source" : "der Eingangspuffer"
TRUE_I : "belegt"
FALSE_I : "leer"

%%15 -BOOL -VAR_INPUT
"Piece_at_drain" : "der Ausgangspuffer"
TRUE_I : "belegt"
FALSE_I : "frei"

%%16 -BOOL -VAR_INPUT
"Em_stopp" : "der Not-Aus-Taster"
FALSE_I : "gedrückt"
TRUE_I : "nicht gedrückt"

%%17 -BOOL -VAR_INPUT
"Restart" : "der ReStart-Taster"
```

```
TRUE_I : "gedrückt"
FALSE_I : "nicht gedrückt"
```

Aktoren des MMI / Anzeigen

| **Technische Bezeichnung** | **Variable** | **Datentyp** | **Kommentar** |
|---|---|---|---|
| Anzeige „Eingangs-zähler“ | Count_in | Integer | Anzahl Teileeingang |
| Anzeige „Ausgangs-zähler“ | Count_out | Integer | Anzahl Teileausgang |
| Anzeige „Anlage bereit“ | LED_Cell_ready | Bool | TRUE - Zelle ist bereit |
| Anzeige „Automatik läuft“ | LED_Cell_run | Bool | TRUE - Zelle läuft automatisch |
| Anzeige „Not-Aus“ | LED_em_stopp | Bool | TRUE - Not-Aus wurde betätigt |

*Tabelle 15 - Datenebene 3: Aktoren des HMI / Anzeigen*

```
%%18 -INT -VAR_OUTPUT
"Count_in" : "der Eingangszähler"
1_O : "um 1 erhöht"
0_O : "auf 0 gesetzt"

%%19 -INT -VAR_OUTPUT
"Count_out" : "der Ausgangszähler"
1_O : "um 1 erhöht"
0_O : "auf 0 gesetzt"

%%20 -BOOL -VAR_OUTPUT
"LED_cell_rdy" : "die Anzeige 'Zelle bereit'"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
TRUE_O : "aktiviert"
FALSE_O : "deaktiviert"

%%21 -BOOL -VAR_OUTPUT
"LED_cell_run" : "die Anzeige 'Automatik läuft'"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
TRUE_O : "aktiviert"
FALSE_O : "deaktiviert"

%%22 -BOOL -VAR_OUTPUT
"LED_em_stopp" : "die Anzeige 'Not-Aus'"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
```

```
TRUE_O : "aktiviert"
FALSE_O : "deaktiviert"
```

*Informelle Spezifikation:*

Jede einzelne Produktionszelle muss die Bereitschafts- sowie eine Fertigmeldung generieren, die der übergeordneten Auftragsverwaltung anzeigen soll, dass die Produktionszelle zur Ausführung eines neuen Auftrags bereit ist bzw. dass ein zuvor gestellter Auftrag fertig bearbeit wurde.

*Definition neuer Daten mit dem SFS-Editor:*

Um die notwendigen Meldungen zu generieren, müssen innerhalb der zentralen Steuerung der Produktionszelle eindeutige Zustände unterschieden werden können, in denen sich die Produktionszelle befinden kann. Diese Zustände sind:

- Ruhezustand, wartend (Bereitschafts- und Fertigmeldung sind gesetzt)
- Automatikbetrieb
- Not-Aus aktiv
- Not-Aus quittiert (Not-Aus-Situation wurde beseitigt, aber noch nicht freigegeben)

Interne Variable des HMI

| **Technische Bezeichnung** | **Variable** | **Datentyp** | **Kommentar** |
|---|---|---|---|
| Zustandsvariable der Zellensteuerung | `cell_1_state` | Integer | |

*Tabelle 16 - Datenebene 3: Interne Variablen des HMI*

```
%%23 -INT -VAR_GLOBAL
"cell_1_state" : "die Produktionszelle 1"
0_I : "im Ruhezustand 0"
1_I : "im Automatikbetrieb"
2_I : "im Not-Aus-Zustand"
3_I : "im Not-Aus-Zustand 2"
0_O : "in den Ruhezustand überführt"
1_O : "in den Automatikzustand überführt"
2_O : "in den Not-Aus-Zustand überführt"
3_O : "in den Not-Aus-Zustand 2 überführt"
```

Darüber hinaus muss eine erkannte Not-Aus-Situation auch an die noch zu definierenden Steuerungen der Einzelkomponenten übertragen werden. Auch hierfür sind spezielle Variablen zu definieren.

Interne Variablen des HMI

| Technische Bezeichnung | Variable | Datentyp | Kommentar |
|---|---|---|---|
| Not-Aus-Signal für die Kransteuerung | Em_stopp_crane | Bool | |
| Not-Aus-Signal für die Steuerung des Zuführbands | Em_stopp_fbelt | Bool | |
| Not-Aus-Signal für die Steuerung des Hubdrehtischs | Em_stopp_table | Bool | |
| Not-Aus-Signal für die Robotersteue-rung | Em_stopp_robot | Bool | |
| Not-Aus-Signal für die Pressensteue-rung | Em_stopp_press | Bool | |
| Not-Aus-Signal für die Steuerung des Ablagebands | Em_stopp_dbelt | Bool | |

*Tabelle 17 - Datenebene 3: Interne Variablen des HMI*

```
%%24 -BOOL -VAR_GLOBAL
"Em_stopp_crane" : "das Not-Aus-Signal für den Kran"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%25 -BOOL -VAR_GLOBAL
"Em_stopp_fbelt" : "das Not-Aus-Signal für das Zuführband"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%26 -BOOL -VAR_GLOBAL
"Em_stopp_table" : "das Not-Aus-Signal für den Hubdrehtisch"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%27 -BOOL -VAR_GLOBAL
"Em_stopp_robot" : "das Not-Aus-Signal für den Roboter"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%28 -BOOL -VAR_GLOBAL
"Em_stopp_press" : "das Not-Aus-Signal für die Presse"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
```

```
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%29 -BOOL -VAR_GLOBAL
"Em_stopp_dbelt" : "das Not-Aus-Signal für das Ablageband"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"
```

Es wird eine Ablaufsteuerung mit einer Zustandsvariable (Integer) aufgebaut. Weiterschaltbedingungen innerhalb dieses Ablaufs sind:

- der Eingangspuffer ist belegt, d. h. es gibt unbearbeitete Teile,
- Not-Aus wurde betätigt,
- Not-Aus wurde wieder entriegelt,
- Re-Start-Taster wurde betätigt.

Die Überführung vom Zustand „Automatikbetrieb“ in den Ruhezustand darf nicht allein durch bloße Überwachung der Belegung des Eingangspuffers mit Rohteilen erfolgen. Dieser Übergang darf vielmehr erst dann erfolgen, wenn alle Werkstücke, die in die Zelle hineingelangt sind, auch wieder herausgekommen sind.

- Überwachung der Differenz zwischen Eingangs- und Ausgangszähler

Interne Variable des HMI

| **Technische Bezeichnung** | **Variable** | **Datentyp** | **Kommentar** |
|---|---|---|---|
| Teiledifferenz | `pieces_diff` | Integer | |

*Tabelle 18 - Datenebene 3: Interne Variablen des HMI*

```
%%30 -INT -VAR_GLOBAL
"pieces_diff" : "die Differenz zwischen Eingangs- und
Ausgangszähler"
0_I : "0"
0_I : "größer als 0"
1_O : "um 1 erhöht"
-1_O : "um 1 verringert"
```

*Informelle Spezifikation:*

Die Produktionszelle befindet sich standardmäßig im Bereitschaftszustand, die Anzeige „Anlage bereit“ soll leuchten. Die Zähleranzeigen für die Anzahl der eingegangenen sowie der ausgegangenen Teile soll auf „0“ stehen. Die Produktionszelle soll komplett selbständig arbeiten. Mit dem Bearbeitungsprozess soll unmittelbar begonnen werden, sobald ein Rohteil im Eingangspuffer liegt (erkennbar durch den Sensor am Eingangspuffer) und keine anderen Bedingungen gegen den Beginn der Bearbeitung sprechen. In diesem Fall soll die Anzeige „Anlage bereit“ verlöschen und die Anzeige „Automatik läuft“ soll aufleuchten.

Durch jedes Teil, das über den Eingangspuffer in die Produktionszelle hineingelangt, soll der Eingangszähler um den Wert „1“ erhöht werden, durch jedes Teil, das die Produktionszelle verlässt (Sensor im Ausgangspuffer), soll der Ausgangszähler um den Wert „1“ erhöht werden. Der automatische Betrieb wird solange aufrechterhalten, wie sich Rohteile im Eingangspuffer befinden. Befinden sich keine Rohteile mehr im Eingangspuffer, werden alle Teile, die sich noch in der Produktionszelle befinden, weiter bearbeitet, bis auch sie die Zelle verlassen haben. In diesem Fall zeigen der Eingangs- und der Ausgangszähler den gleichen Wert an, d. h. die Bearbeitung dieser Charge wurde vollständig abgeschlossen. Dadurch werden die Fertig- und die Bereitschaftsmeldung der Produktionszelle gesetzt, die Anzeige „Automatik läuft“ erlischt und die Anzeige „Anlage bereit“ leuchtet auf.

Tritt während des automatischen Betriebs der Produktionszelle eine Störung auf, so kann, z. B. durch das Bedienpersonal, ein Not-Aus ausgelöst werden. In diesem Fall müssen alle Aktionen innerhalb der Produktionszelle gestoppt werden, d. h. alle Aktoren, bis auf die Magnetgreifer; werden auf FALSE gesetzt. Die Magnetgreifer behalten aus Sicherheitsgründen ihren aktuellen Zustand bei. Gleichzeitig muss die Anzeige „Automatik läuft“ verlöschen, die Anzeige „Not-Aus“ muss aufleuchten. Nachdem die gefährliche Situation beseitigt und der Not-Aus-Taster entriegelt wurde, kann der automatische Betrieb der Produktionszelle durch Betätigen des Tasters „Re-Start“ wieder aufgenommen werden ***(dieses Verhalten entspricht der Stop-Kategorie I für eine Not-Aus-Funktion gemäß DIN EN 418.)***.

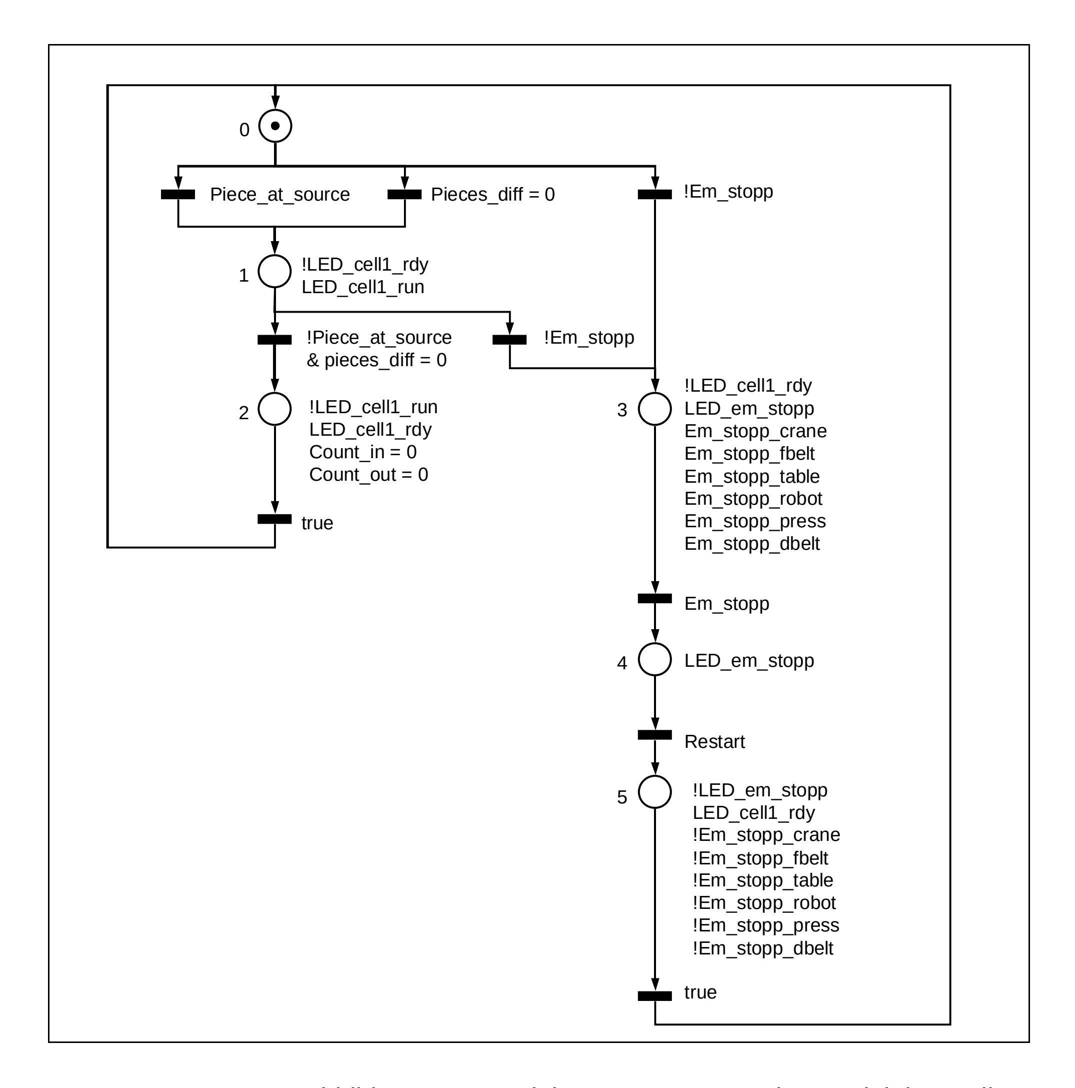



*Abbildung 30 - Betriebsartensteuerung der Produktionszelle*

*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 5<<< (einfache Forderung - direkt) */*

*Wenn "die Produktionszelle 1" "im Ruhezustand 0" ist und "der Eingangspuffer" ist "belegt" , dann muss "die Produktionszelle 1" unmittelbar "in den Automatikzustand überführt" werden und "die Anzeige 'Zelle ist bereit'" muss unmittelbar "deaktiviert" werden und "die Anzeige 'Automatik läuft'" muss unmittelbar "aktiviert" werden .*

```
AG ( (rdy_in & cell_1_state=0 & Piece_at_source) ->
A[!rdy_plc U (rdy_plc & cell_1_state=1 & !LED_cell1_rdy &
LED_cell1_run) ] )
```

*/* >>>Satz 6<<< (einfache Forderung - direkt) */*

*Wenn "die Produktionszelle 1" "im Ruhezustand 0" ist und "die Differenz zwischen Eingangs- und Ausgangszähler" ist "größer als 0" , dann muss "die Produktionszelle 1" unmittelbar "in den Automatikzustand überführt" werden und "die Anzeige 'Zelle ist bereit'" muss unmittelbar "deaktiviert" werden und "die Anzeige 'Automatik läuft'" muss unmittelbar "aktiviert" werden .*

```
AG ( (rdy_in & cell_1_state=0 & pieces_diff=0) ->
A[!rdy_plc U (rdy_plc & cell_1_state=1 & !LED_cell1_rdy &
LED_cell1_run) ] )
```

*/* >>>Satz 7<<< (einfache Forderung - direkt) */*

*Wenn "die Produktionszelle 1" "im Ruhezustand 0" ist und "der Not-Aus-Taster" ist "gedrückt" , dann muss "die Produktionszelle 1" unmittelbar "in den Not-Aus-Zustand überführt" werden und "die Anzeige 'Zelle ist bereit'" muss unmittelbar "deaktiviert" werden und "die Anzeige 'Not-Aus'" muss unmittelbar "aktiviert" werden und "das Not-Aus-Signal für den Kran" muss unmittelbar "gesetzt" werden und "das Not-Aus-Signal für das Zuführband" muss unmittelbar "gesetzt" werden und "das Not-Aus-Signal für den Hubdrehtisch" muss unmittelbar "gesetzt" werden und "das Not-Aus-Signal für den Roboter" muss unmittelbar "gesetzt" werden und "das Not-Aus-Signal für die Presse" muss unmittelbar "gesetzt" werden und "das Not-Aus-Signal für das Ablageband" muss unmittelbar "gesetzt" werden .*

```
AG ( (rdy_in & cell_1_state=0 & !Em_stopp) -> A[!rdy_plc U
(rdy_plc & cell_1_state=3 & !LED_cell1_rdy & LED_em_stopp
& Em_stopp_crane & Em_stopp_fbelt & Em_stopp_table &
Em_stopp_robot & Em_stopp_press & Em_stopp_dbelt) ] )
```

*/* >>>Satz 8<<< (einfache Forderung - direkt) */*

*Wenn "die Produktionszelle 1" "im Automatikbetrieb" ist und "der Eingangspuffer" ist "leer" und "die Differenz zwischen Eingangs- und Ausgangszähler" ist "0" , dann muss "die Produktionszelle 1" unmittelbar "in den Ruhezustand überführt" werden und "die Anzeige 'Automatik läuft'" muss unmittelbar "deaktiviert" werden und "die Anzeige 'Zelle ist bereit'" muss unmittelbar "aktiviert" werden und "der Eingangszähler" muss unmittelbar "auf 0 gesetzt" werden und "der Ausgangszähler" muss unmittelbar "auf 0 gesetzt" werden .*

```
AG ( (rdy_in & cell_1_state=1 & !Piece_at_source &
pieces_diff=0) -> A[!rdy_plc U (rdy_plc & cell_1_state=0 &
!LED_cell1_run & LED_cell1_rdy & Count_in=0 & Count_out=0)
] )
```

*/* >>>Satz 9<<< (einfache Forderung - direkt) */*

*Wenn "die Produktionszelle 1" "im Automatikbetrieb" ist und "der Not-Aus-Taster" ist "gedrückt" , dann muss "die Produktionszelle 1" unmittelbar "in den Not-Aus-Zustand überführt" werden und "die Anzeige 'Automatik läuft'" muss unmittelbar "deaktiviert" werden und "die Anzeige 'Not-Aus'" muss unmittelbar "aktiviert" werden und "das Not-Aus-Signal für den Kran" muss unmittelbar "gesetzt" werden und "das Not-Aus-Signal für das Zuführband" muss unmittelbar "gesetzt" werden und "das Not-Aus-Signal für den Hubdrehtisch" muss unmittelbar "gesetzt" werden und "das Not-Aus-Signal für den Roboter" muss unmittelbar "gesetzt" werden und "das Not-Aus-Signal für die Presse" muss unmittelbar "gesetzt" werden und "das Not-Aus-Signal für das Ablageband" muss unmittelbar "gesetzt" werden .*

```
AG ( (rdy_in & cell_1_state=1 & !Em_stopp) -> A[!rdy_plc U
(rdy_plc & cell_1_state=3 & !LED_cell1_run & LED_em_stopp
& Em_stopp_crane & Em_stopp_fbelt & Em_stopp_table &
Em_stopp_robot & Em_stopp_press & Em_stopp_dbelt) ] )
```

*/* >>>Satz 10<<< (einfache Forderung - direkt) */*

*Wenn "die Produktionszelle 1" "im Not-Aus-Zustand" ist und "der Not-Aus-Taster" ist "nicht gedrückt" , dann muss "die Produktionszelle 1" unmittelbar "in den Not-Aus-Zustand 2 überführt" werden und "die Anzeige 'Not-Aus'" muss unmittelbar "aktiviert" werden .*

```
AG ( (rdy_in & cell_1_state=2 & Em_stopp) -> A[!rdy_plc U
(rdy_plc & cell_1_state=4 & LED_em_stopp) ] )
```

*/* >>>Satz 11<<< (einfache Forderung - direkt) */*

*Wenn "die Produktionszelle 1" "im Not-Aus-Zustand 2" ist und "der ReStart-Taster" ist "gedrückt" , dann muss "die Produktionszelle 1" unmittelbar "in den Ruhezustand überführt" werden und "die Anzeige 'Not-Aus'" muss unmittelbar "deaktiviert" werden und "die Anzeige 'Zelle ist bereit'" muss unmittelbar "aktiviert" werden und "das Not-Aus-Signal für den Kran" muss unmittelbar "zurückgesetzt" werden und "das Not-Aus-Signal für das Zuführband" muss unmittelbar "zurückgesetzt" werden und "das Not-Aus-Signal für den Hubdrehtisch" muss unmittelbar "zurückgesetzt" werden und "das Not-Aus-Signal für den Roboter" muss unmittelbar "zurückgesetzt" werden und "das Not-Aus-Signal für die Presse" muss unmittelbar "zurückgesetzt" werden und "das Not-Aus-Signal für das Ablageband" muss unmittelbar "zurückgesetzt" werden .*

```
AG ( (rdy_in & cell_1_state=4 & Restart) -> A[!rdy_plc U
(rdy_plc & cell_1_state=0 & !LED_em_stopp & LED_cell1_rdy
& !Em_stopp_crane & !Em_stopp_fbelt & !Em_stopp_table &
!Em_stopp_robot & !Em_stopp_press & !Em_stopp_dbelt) ] )
```

*/* >>>Satz 12<<< (einfache Forderung - direkt) */*

*Wenn "der Eingangspuffer" "belegt" ist , dann muss "die Differenz zwischen Eingangs- und Ausgangszähler" unmittelbar "um 1 erhöht" werden .*

```
AG ( (rdy_in & Piece_at_source) -> A[!rdy_plc U (rdy_plc &
pieces_diff=1) ] )
```

*/* >>>Satz 13<<< (einfache Forderung - direkt) */*

*Wenn "der Eingangspuffer" "belegt" ist , dann muss "der Eingangszähler" unmittelbar "um 1 erhöht" werden .*

```
AG ( (rdy_in & Piece_at_source) -> A[!rdy_plc U (rdy_plc &
Count_in=1) ] )
```

*/* >>>Satz 14<<< (einfache Forderung - direkt) */*

*Wenn "der Ausgangspuffer" "belegt" ist , dann muss "die Differenz zwischen Eingangs- und Ausgangszähler" unmittelbar "um 1 verringert" werden .*

```
AG ( (rdy_in & Piece_at_drain) -> A[!rdy_plc U (rdy_plc &
pieces_diff=-1) ] )
```

*/* >>>Satz 15<<< (einfache Forderung - direkt) */*

*Wenn "der Ausgangspuffer" "belegt" ist , dann muss "der Ausgangszähler" unmittelbar "um 1 erhöht" werden .*

```
AG ( (rdy_in & Piece_at_drain) -> A[!rdy_plc U (rdy_plc &
Count_out=1) ] )
```

*Informelle Spezifikation weiterer Eigenschaften*

Ein von der zentralen Steuerung erkanntes Not-Aus-Signal muss unverzüglich an die untergeordneten Steuerungen der Einzelkomponenten weitergeleitet werden.

*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 16<<< (einfache Forderung - direkt) */*

*Wenn "der Not-Aus-Taster" "gedrückt" ist , dann muss unmittelbar "das Not-Aus-Signal für das Ablageband" "gesetzt" werden und "das Not-Aus-Signal für das Zuführband" muss unmittelbar "gesetzt" werden und "das Not-Aus-Signal für den Hubdrehtisch" muss unmittelbar "gesetzt" werden und "das Not-Aus-Signal für den Kran" muss unmittelbar "gesetzt" werden und "das Not-Aus-Signal für den Roboter" muss unmittelbar "gesetzt" werden und "das Not-Aus-Signal für die Presse" muss unmittelbar "gesetzt" werden .*

```
AG ( (rdy_in & !Em_stopp) -> A[!rdy_plc U (rdy_plc &
Em_stopp_dbelt & Em_stopp_fbelt & Em_stopp_table &
Em_stopp_crane & Em_stopp_robot & Em_stopp_press) ] )
```

*/* >>>Satz 17<<< (einfache Forderung - Zustand) */*

*Wenn "der Not-Aus-Taster" "gedrückt" ist , dann muss gleichzeitig "das Not-Aus-Signal für das Ablageband" "gesetzt" sein und "das Not-Aus-Signal für das Zuführband" muss gleichzeitig "gesetzt" sein und "das Not-Aus-Signal für den Hubdrehtisch" muss gleichzeitig "zurückgesetzt" sein und "das Not-Aus-Signal für den Kran" muss gleichzeitig "gesetzt" sein und "das Not-Aus-Signal für den Roboter" muss gleichzeitig "gesetzt" sein und "das Not-Aus-Signal für die Presse" muss gleichzeitig "gesetzt" sein .*

```
AG ( (rdy_plc & !Em_stopp) -> (Em_stopp_dbelt &
Em_stopp_fbelt & !Em_stopp_table & Em_stopp_crane &
```

```
Em_stopp_robot & Em_stopp_press) )
```

### A.7.2.2 Kommunikation mit dem Kran (I)

*Informelle Spezifikation:*

Die Steuerung der Produktionszelle muss an die Steuerung des Krans ein Signal geben, wenn dieser ein Rohteil aus dem Eingangspuffer holen soll. Dazu müssen der Zellensteuerung die folgenden Informationen bekannt sein:

- im Eingangspuffer liegt ein Rohteil,
- das Rohteil kann auf dem Zuführband abgelegt werden.

*Definition neuer Daten mit dem SFS-Editor:*

Während die Variable für die Belegung des Eingangspuffers bereits definiert ist, müssen weitere Variablen noch definiert werden. So muss das Zuführband eine Meldung generieren, dass es zur Übernahme eines neuen Teils bereit ist. Weiterhin wird ein Startsignal definiert, das den Kran zum Transport eines Rohteils vom Eingangspuffer zum Zuführband veranlasst. Der Kran wiederum muss mit einer weiteren Variablen die erfolgreiche Ausführung des Transports an die zentrale Steuerung weiterleiten.

| **Technische Bezeichnung** | **Variable** | **Datentyp** | **Kommentar** |
|---|---|---|---|
| Meldung „Zuführband für Aufnahme eines Rohteils bereit“ | `Fbelt_rdy_take` | Bool | |
| Startsignal „Rohteil holen“ | `Crane_run_source` | Bool | |
| Fertigmeldung Kran | `Crane_done_give_fbelt` | Bool | Kran hat Rohteil auf Zuführband abgelegt |

```
%%31 -BOOL -VAR_GLOBAL
"Fbelt_rdy_take" : "die Bereitschaftsmeldung 'Zuführband ist
zur Übernahme eines Teils bereit'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%32 -BOOL -VAR_GLOBAL
"Crane_run_source" : "das Startsignal 'Kran holt Rohteil vom
Eingangspuffer'"
TRUE_I : "gesetzt"
```

```
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%33 -BOOL -VAR_GLOBAL
"Crane_done_give_fbelt" : "die Fertigmeldung 'Kran hat ein
Rohteil auf das Zuführband gelegt'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"
```

*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 23<<< (einfache Forderung - direkt) */*

*Wenn "die Produktionszelle 1" "im Automatikbetrieb" ist und "der Eingangspuffer" ist "belegt" und "die Bereitschaftsmeldung 'Zuführband ist zur Übernahme eines Teils bereit'" ist "gesetzt" , dann muss "die Bereitschaftsmeldung 'Zuführband ist zur Übernahme eines Teils bereit'" unmittelbar "zurückgesetzt" werden und "das Startsignal 'Kran holt Rohteil von Eingangspuffer'" muss unmittelbar "gesetzt" werden .*

```
AG ( (rdy_in & cell_1_state=1 & Piece_at_source &
Fbelt_rdy_take) -> A[!rdy_plc U (rdy_plc & !Fbelt_rdy_take
& Crane_run_source) ] )
```

*Weitere informelle Spezifikationen:*

Es darf kein Start erfolgen, wenn der Eingangspuffer nicht belegt ist.

Es darf kein Start erfolgen, wenn das Zuführband nicht bereit ist.

*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 24<<< (erweiterte Möglichkeit - Zustand) */*

*"das Startsignal 'Kran holt Rohteil vom Eingangspuffer'" darf nur "gesetzt" sein , wenn gleichzeitig "der Eingangspuffer" "belegt" ist .*

```
AG !( rdy_plc & !(Piece_at_source) & (Crane_run_source) )
```

*/* >>>Satz 25<<< (erweiterte Möglichkeit - Zustand) */*

*"das Startsignal 'Kran holt Rohteil vom Eingangspuffer'" darf nur "gesetzt" sein , wenn gleichzeitig "die Bereitschaftsmeldung 'Zuführband ist zur Übernahme eines Teils bereit'" "gesetzt" ist .*

```
AG !( rdy_plc & !(Fbelt_ready_take) & (Crane_run_source) )
```

### *A.7.2.3 Kommunikation mit Zuführband*

*Vorüberlegung:*

Das Zuführband dient dazu, die Rohteile, die der Kran auf dem Zuführband abgelegt hat, auf den Hubdrehtisch zu befördern.

Das Zuführband kann nicht selbständig erkennen, wenn ein Rohteil an seinem Bandanfang abgelegt wurde, es benötigt also von der Zellensteuerung ein Signal, wenn es mit dem Transport des Rohteils beginnen soll. Dieser Transport kann aber erst erfolgen, wenn der Hubdrehtisch auch zur Aufnahme eines Teils bereit (also leer und korrekt positioniert) ist.

Aus Effizienzgründen kann man den Transportvorgang in zwei Phasen unterteilen. So kann in der ersten Phase das Rohteil bereits zum Bandende transportiert werden, auch wenn der Hubdrehtisch noch nicht zur Aufnahme des Teils bereit ist. Erst wenn dieser Zustand erreicht ist, findet in der zweiten Transportphase die Übergabe des Rohteils auf den Hubdrehtisch statt. Dieses Vorgehen hat den weiteren Vorteil, dass das Zuführband bereits wieder mit einem neuen Rohteil beladen werden kann, während noch auf die Bereitschaft des Hubdrehtischs gewartet wird. Für diesen Zeitraum würden sich dann zwei Rohteile auf dem Band befinden.

*Informelle Spezifikation:*

Der Transportvorgang wird in zwei separate Phasen mit eigenen Startbedingungen unterteilt.

1. Transport des Rohteils bis zum Bandende,
2. Übergabe des Rohteils auf den Hubdrehtisch.

zu 1. Durch den Kran wird bereits eine Meldung zur Verfügung gestellt, dass er das Zuführband beladen hat. Das Zuführband muss seinerseits eine Meldung zur Verfügung stellen, dass es zum Weitertransport dieses Teils bereit ist (das Ende des Zuführbands muss frei sein). Außerdem ist ein Startsignal für diesen Transport zu generieren.

zu 2. Durch den Hubdrehtisch muss eine Meldung zur Verfügung gestellt werden, dass er zur Annahme eines Teils bereit ist (wenn er also in der richtigen Höhe und Position ist), das Zuführband muss seinerseits eine Meldung zur Verfügung stellen, dass es zur Übergabe eines Teils bereit ist (Bandende muss belegt sein). Außerdem ist ein Startsignal für diese Übergabe zu generieren.

*Definition neuer Daten mit dem SFS-Editor:*

| **Technische Bezeichnung** | **Variable** | **Datentyp** | **Kommentar** |
|---|---|---|---|
| Bereitschaftsmeldung „Zuführband bereit zum Transport“ | `Fbelt_ready_transport` | Bool | Zuführband ist zum Transport bereit |
| Startsignal „Rohteil zum | `Fbelt_run_transport` | Bool | Zuführband soll |

| | | | |
|---|---|---|---|
| Bandende transportieren“ | | | Teil zum Bandende transportieren |
| Bereitschaftsmeldung „Zuführband bereit zur Übergabe“ | `Fbelt_ready_give` | Bool | Zuführband ist zur Übergabe bereit |
| Fertigmeldung „Tisch bereit zur Übernahme“ | `Table_ready_take` | Bool | Tisch ist zu Entgegennahme eines Teils bereit |
| Startsignal „Rohteil zum Hubdrehtisch transportieren“ | `Fbelt_run_table` | Bool | Zuführband soll Teil auf Tisch transportieren |
| Übergabemeldung des Zuführbands | `Fbelt_done_give` | Bool | Zuführband hat Teil auf Tisch transportiert |

*Tabelle 19 - Datenebene 3: Variablen zur Kommunikation mit dem Zuführband*

```
%%34 -BOOL -VAR_GLOBAL
"Fbelt_ready_transport" : "die Bereitschaftsmeldung
'Zuführband ist zum Transport bereit'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%35 -BOOL -VAR_GLOBAL
"Fbelt_run_transport" : "das Startsignal 'Zuführband soll Teil
zum Bandende transportieren'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%36 -BOOL -VAR_GLOBAL
"Fbelt_ready_give" : "die Bereitschaftsmeldung 'Zuführband ist
zur Übergabe bereit'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%37 -BOOL -VAR_GLOBAL
"Table_ready_take" : "die Bereitschaftsmeldung 'Hubdrehtisch
ist zur Übernahme eines Teils bereit'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"
```

```
%%38 -BOOL -VAR_GLOBAL
"Fbelt_run_table" : "das Startsignal 'Zuführband soll Teil auf
Tisch befördern'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%39 -BOOL -VAR_GLOBAL
"Fbelt_done_give" : "die Fertigmeldung 'Zuführband hat Teil
auf Tisch abgelegt'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"
```

*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 27<<< (einfache Forderung - direkt) */*

*Wenn "die Produktionszelle 1" "im Automatikbetrieb" ist und "die Fertigmeldung 'Kran hat eine Rohteil auf das Zuführband gelegt'" ist "gesetzt" und "die Bereitschaftsmeldung 'Zuführband ist zum Transport bereit'" ist "gesetzt" , dann muss "die Fertigmeldung 'Kran hat eine Rohteil auf das Zuführband gelegt'" unmittelbar "zurückgesetzt" werden und muss "die Bereitschaftsmeldung 'Zuführband ist zum Transport bereit'" unmittelbar "zurückgesetzt" werden und muss "das Startsignal 'Zuführband soll Teil zum Bandende transportieren'" unmittelbar "gesetzt" werden .*

```
AG ( (rdy_in & cell_1_state=1 & Crane_done_give_fbelt &
Fbelt_rdy_transport) -> A[!rdy_plc U (rdy_plc &
!Crane_done_give_fbelt & !Fbelt_rdy_transport &
Fbelt_run_transport) ] )
```

*/* >>>Satz 28<<< (einfache Forderung - direkt) */*

*Wenn "die Produktionszelle 1" "im Automatikbetrieb" ist und "die Bereitschaftsmeldung 'Hubdrehtisch ist zur Übernahme eines Teils bereit'" ist "gesetzt" und "die Bereitschaftsmeldung 'Zuführband ist zur Übergabe bereit'" ist "gesetzt" , dann muss "die Bereitschaftsmeldung 'Hubdrehtisch ist zur Übernahme eines Teils bereit'" unmittelbar "zurückgesetzt" werden und muss "die Bereitschaftsmeldung 'Zuführband ist zur Übergabe bereit'" unmittelbar "zurückgesetzt" werden und "das Startsignal 'Zuführband soll Teil auf Tisch befördern'" muss unmittelbar "gesetzt" werden .*

```
AG ( (rdy_in & cell_1_state=1 & Table_rdy_take &
Fbelt_rdy_give) -> A[!rdy_plc U (rdy_plc & !Table_rdy_take
& !Fbelt_rdy_give & Fbelt_run_table) ] )
```

*Informelle Spezifikation weiterer Eigenschaften:*

Es darf kein Transport erfolgen, wenn der Kran kein Rohteil abgelegt hat.

Es darf kein Transport erfolgen, wenn das Zuführband nicht zum Transport bereit ist.

Es darf keine Übergabe erfolgen, wenn der Hubdrehtisch nicht bereit ist.

Es darf keine Übergabe erfolgen, wenn das Zuführband nicht zur Übergabe bereit ist.

*Formale Spezifikation mit dem SFS-Editor*

*/* >>>Satz 29<<< (einfaches Verbot - Zustand) */*

*"das Startsignal 'Zuführband soll Teil zum Bandende transportieren'" darf nicht gleichzeitig "gesetzt" sein , wenn "die Fertigmeldung 'Kran hat ein Rohteil auf das Zuführband gelegt'" "nicht gesetzt" ist .*

```
AG !( rdy_plc & !Crane_done_give_fbelt &
(Fbelt_run_transport) )
```

*/* >>>Satz 30<<< (einfaches Verbot - Zustand) */*

*"das Startsignal 'Zuführband soll Teil zum Bandende transportieren'" darf nicht gleichzeitig "gesetzt" sein , wenn "die Bereitschaftsmeldung 'Zuführband ist zum Transport bereit'" "nicht gesetzt" ist .*

```
AG !( rdy_plc & !Fbelt_ready_transport &
(Fbelt_run_transport) )
```

*/* >>>Satz 31<<< (einfaches Verbot - Zustand) */*

*"das Startsignal 'Zuführband soll Teil auf Tisch befördern'" darf nicht gleichzeitig "gesetzt" sein , wenn "die Bereitschaftsmeldung 'Hubdrehtisch ist zur Übernahme eines Teils bereit'" "nicht gesetzt" ist .*

```
AG !( rdy_plc & !Table_ready_take & (Fbelt_run_table) )
```

*/* >>>Satz 32<<< (einfaches Verbot - Zustand) */*

*"das Startsignal 'Zuführband soll Teil auf Tisch befördern'" darf nicht gleichzeitig "gesetzt" sein , wenn "die Bereitschaftsmeldung 'Zuführband ist zur Übergabe bereit'" "nicht gesetzt" ist .*

```
AG !( rdy_plc & !Fbelt_ready_give & (Fbelt_run_table) )
```

#### *A.7.2.4 Kommunikation mit dem Hubdrehtisch*

*Informelle Spezifikation:*

Der Hubdrehtisch dient dazu, die Rohteile, die durch das Zuführband auf dem Hubdrehtisch abgelegt wurden, in eine Position zu bringen, von der sie durch den Roboter übernommen werden können. Nachdem der Roboter das Rohteil vom Hubdrehtisch entnommen hat, muss sich dieser wieder in seine Ausgangsposition zurückbewegen.

Der Hubdrehtisch kann nicht selbst erkennen, wenn er mit einem Teil beladen wurde, er muss deshalb von der zentralen Steuerung ein Startsignal bekommen. Außerdem benötigt er ebenso ein Startsignal, wann der Roboter das Rohteil wieder vom Tisch genommen hat, und er sich wieder in seine Ausgangsposition bewegen kann.

Auch der Transportvorgang durch den Hubdrehtisch kann wieder in zwei Phasen unterteilt werden:

1. Anheben und Drehen des beladenen Tisches
2. Absenken und Zurückdrehen des leeren Tisches

*Definition neuer Daten mit dem SFS-Editor*

Die Meldung, dass das Zuführband ein Rohteil auf dem Tisch abgelegt hat, existiert bereits. Es wird ein Signal definiert, dass die Positionierung gestartet werden kann. Der Hubdrehtisch muss die Beendigung der Positionierung melden.

Es wird eine Meldung benötigt, dass der Roboter das Teil vom Hubdrehtisch genommen hat. Danach kann die Rückbewegung gestartet werden. Der Hubdrehtisch muss auch die Beendigung dieser Aktion bestätigen (die Variable `Table_ready_take` ist bereits definiert).

Variablen zur Kommunikation mit dem Hubdrehtisch

| **Technische Bezeichnung** | **Variable** | **Datentyp** | **Kommentar** |
|---|---|---|---|
| Startsignal „Heben/Drehen“ | `Table_run_lift` | Bool | |
| Meldung „Hubdrehtisch ist zur Übergabe bereit“ | `Table_ready_give` | Bool | |
| Fertigmeldung „Roboter hat Teil vom Tisch genommen“ | `Robot_done_take_table` | Bool | |
| Startsignal „Senken/Drehen“ | `Table_run_lower` | Bool | |

*Tabelle 20 - Datenebene 3: Variablen zur Kommunikation mit dem Hubdrehtisch*

```
%%40 -BOOL -VAR_GLOBAL
"Table_run_lift" : "das Startsignal 'Hubdrehtisch heben und
links drehen'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%41 -BOOL -VAR_GLOBAL
"Table_rdy_give" : "die Bereitschaftsmeldung 'Hubdrehtisch ist
zur Übergabe eines Teils bereit"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%42 -BOOL -VAR_GLOBAL
"Robot_done_take_table" : "die Fertigmeldung 'Roboter hat Teil
vom Tisch genommen'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%43 -BOOL -VAR_GLOBAL
"Table_run_lower" : "das Startsignal 'Hubdrehtisch senken und
rechts drehen'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"
```

*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 34<<< (einfache Forderung - direkt) */*

*Wenn "die Produktionszelle 1" "im Automatikbetrieb" ist und "die Fertigmeldung 'Zuführband hat Teil auf Tisch abgelegt'" ist "gesetzt" , dann muss "die Fertigmeldung 'Zuführband hat Teil auf Tisch abgelegt'" unmittelbar "zurückgesetzt" werden und "das Startsignal 'Hubdrehtisch heben und links drehen'" muss unmittelbar "gesetzt" werden .*

```
AG ( (rdy_in & cell_1_state=1 & Fbelt_done_give) ->
A[!rdy_plc U (rdy_plc & !Fbelt_done_give & Table_run_lift)
] )
```

*/* >>>Satz 35<<< (einfache Forderung - direkt) */*

*Wenn "die Produktionszelle 1" "im Automatikbetrieb" ist und "die Fertigmeldung 'Roboter hat Teil vom Tisch genommen'" ist "gesetzt" , dann muss "die Fertigmeldung*

*'Roboter hat Teil vom Tisch genommen'" unmittelbar "zurückgesetzt" werden und "das Startsignal 'Hubdrehtisch senken und rechts drehen'" muss unmittelbar "gesetzt" werden .*

```
AG ( (rdy_in & cell_1_state=1 & Robot_done_take_table) ->
A[!rdy_plc U (rdy_plc & !Robot_done_take_table &
Table_run_lower) ] )
```

*Informelle Spezifikation weiterer Eigenschaften:*

Der Hubdrehtisch darf sich nicht heben, wenn das Zuführband nicht übergeben hat.

Der Hubdrehtisch darf sich nicht senken, wenn der Roboter das Teil nicht entnommen hat.

*Formale Spezifikation mit dem SFS-Editor*

*/* >>>Satz 36<<< (einfaches Verbot - Zustand) */*

*"das Startsignal 'Hubdrehtisch heben und links drehen'" darf nicht gleichzeitig "gesetzt" sein , wenn "die Fertigmeldung 'Zuführband hat Teil auf Tisch abgelegt'" "nicht gesetzt" ist .*

```
AG !( rdy_plc & !Fbelt_done_give & (Table_run_lift) )
```

*/* >>>Satz 37<<< (einfaches Verbot - Zustand) */*

*"das Startsignal 'Hubdrehtisch senken und rechts drehen'" darf nicht gleichzeitig "gesetzt" sein , wenn "die Fertigmeldung 'Roboter hat Teil vom Tisch genommen'" "nicht gesetzt" ist .*

```
AG !( rdy_plc & !Robot_done_take_table & (Table_run_lower)
)
```

### A.7.2.5 Kommunikation mit dem Roboter

Der Roboter hat in der Produktionszelle zwei Aufgaben:

- Transport von Rohteilen vom Hubdrehtisch zur Presse,
- Transport von Fertigteilen von der Presse zum Ablageband.

Für jede dieser Aufgaben ist genau ein Arm des Roboters zuständig. Die Beladung der Presse erfolgt mit dem Roboterarm 1, die Entladung mit dem Roboterarm 2. Zur Steigerung der Effizienz der Anlage sind die dargestellten Vorgänge nicht nur alternativ, sondern (unter Berücksichtigung bestimmter Koordinierungsbedingungen) auch gleichzeitig möglich.

Es ist sinnvoll, die Bestückung der Presse und die Entnahme von Fertigteilen aus der Presse zu koordinieren (Abbildung 31). So kann der Roboterarm 2 ein Teil von der Presse zum Ablageband befördern, während der Roboterarm 1 gleichzeitig (nur unterbrochen durch die Bewegungsvorgänge des Arms 2) ein Teil vom Hubdrehtisch zur Presse befördert. Auf Grund der variablen Anzahl der Werkstücke muss diese Steuerung des Roboters dann jedoch so flexibel ausgelegt werden, das verschiedene (von der Anzahl der Werkstücke abhängige) Verhaltensmodi abgearbeitet werden können.

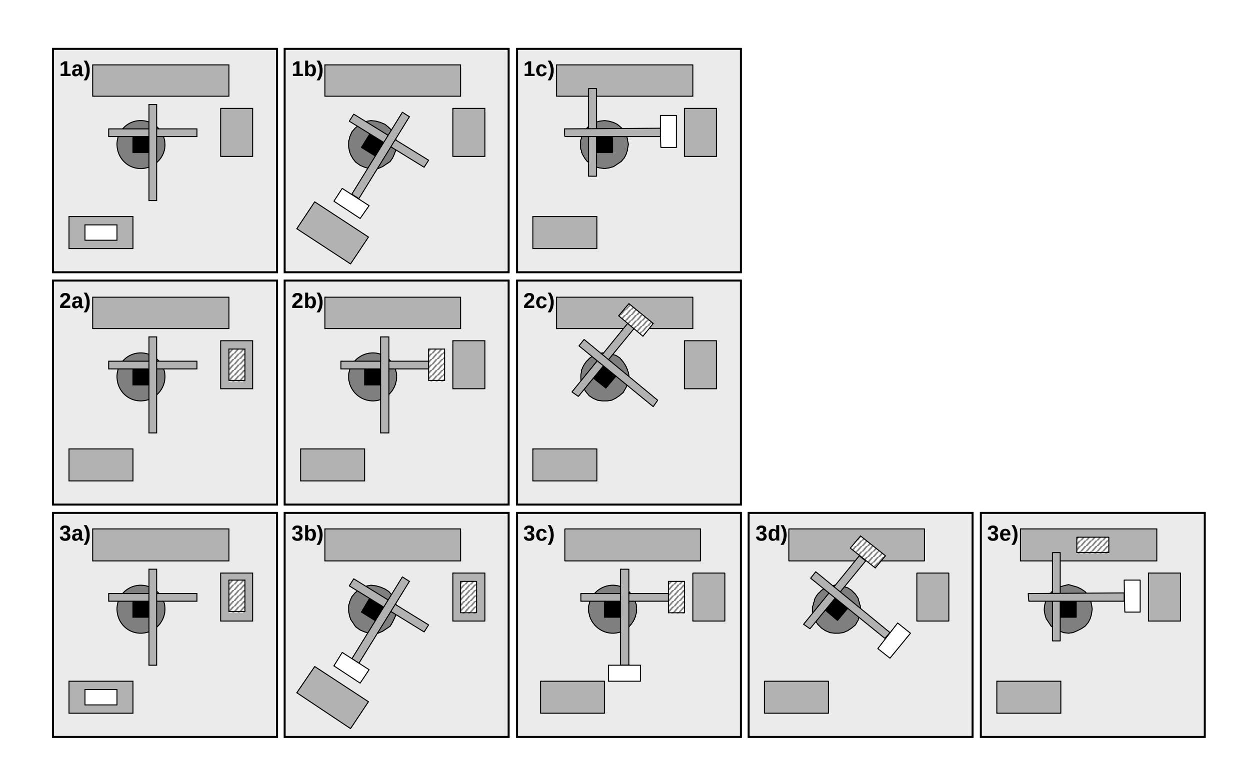



*Abbildung 31 - Arbeitsmodi des Roboters*

Variante 1: es liegt ein Teil auf dem Hubdrehtisch, die Presse und das Ablageband sind leer (Darstellung 1a) → Arm 1 transportiert das Teil in die Presse (1b, 1c), weiter mit Variante 2.

Variante 2: es liegt ein bearbeitetes Teil in der Presse, der Hubdrehtisch und das Ablageband sind leer (Darstellung 2a) → Arm 2 transportiert das Teil auf das Ablageband (2b, 2c),

Variante 3: es liegt ein Teil auf dem Hubdrehtisch und es liegt ein bearbeitetes Teil in der Presse, das Ablageband ist leer (Darstellung 3a) → Arm 1 nimmt das Teil vom Hubdrehtisch auf (3b), Arm 2 entnimmt das bearbeitete Teil aus der Presse (3c), Arm 2 legt das fertige Teil auf dem Ablageband ab (3d), Arm 1 legt das neue Teil in die Presse ein (3e),

*Definition neuer Daten mit dem SFS-Editor*

Der Roboter soll einerseits Rohteile vom Hubdrehtisch zur Presse befördern und andererseits Fertigteile aus der Presse entnehmen und auf dem Ablageband ablegen. Da es wiederum keine Sensoren zur direkten Bestimmung des Vorhandenseins von Werkstücken gibt, benötigt die Robotersteuerung wieder zusätzliche Signale, wann sie mit ihrer Arbeit beginnen soll. Für den Beschickungsvorgang der Presse bedeutet dies, dass es sowohl eine Meldung geben muss, wann der Hubdrehtisch mit einem Rohteil in Position ist (ist bereits definiert), und dass es auch eine Meldung geben muss, wann die Presse zur Aufnahme des Rohteils bereit ist. Für die Entnahme des Fertigteils aus der Presse muss es einerseits eine Meldung geben, wann die Presse mit dem Bearbeitungsvorgang fertig und somit zur Übergabe des Teils bereit ist und wann das Ablageband zur Übernahme des Fertigteils bereit ist.

Wie bereits dargestellt, gibt es insgesamt drei Betriebsmodi des Roboters (vergleiche Abbildung 31), deren Start auf unterschiedliche Bedingungen zurückgeführt werden muss.

1. gleichzeitiges Ent- und Beladen der Presse.
2. nur Beladen der Presse,
3. nur Entladen der Presse,

zu 1. Zunächst wird eine Meldung des Hubdrehtischs benötigt, dass er zur Übergabe eines Rohteils bereit ist, ebenso wird eine Meldung der Presse benötigt, dass sie zur Übergabe eines Fertigteils bereit ist. Letztendlich wird noch eine Meldung des Ablagebands benötigt, dass es zur Übernahme des Fertigteils bereit ist. Daraus kann ein Startsignal für den Mix-Modus generiert werden. Zusätzlich muss beim Wiederbeladen der Presse die Koordination mit der Pressentätigkeit beachtet werden. Der Roboter muss also mitteilen, dass er zum erneuten Beladen der Presse bereit ist. Wenn die Presse dann ihrerseits die Beladestellung erreicht hat, kann der Vorgang fortgesetzt werden.

zu 2. und 3. Es werden wiederum die bereits dargestellten Bereitschaftsmeldungen benötigt (jedoch werden diesmal andere Wahrheitswerte abgefragt), zusätzlich wird für die ausschließliche Beladung der Presse eine Meldung zur Übernahmebereitschaft benötigt. Für den Lade- bzw. Entladevorgang werden separate Startsignale vereinbart.

Variablen zur Kommunikation mit dem Roboter

| Technische Bezeichnung | Variable | Datentyp | Kommentar |
|---|---|---|---|
| Meldung „Presse bereit zur Übernahme eines Rohteils“ | `Press_rdy_take` | Bool | |
| Startsignal „Presse beladen“ | `Robot_run_load` | Bool | |
| Meldung „Presse bereit zur Übergabe eines Fertigteils“ | `Press_rdy_give` | Bool | |
| Meldung „Ablageband fertig zur Übernahme eines Teils“ | `Dbelt_rdy_take` | Bool | |
| Startsignal „Presse entladen“ | `Robot_run_unload` | Bool | |
| Startsignal „Mixbetrieb“ | `Robot_run_mix` | Bool | |
| Meldung „Roboter wartet vor Presse“ | `Robot_wait_mix` | Bool | |
| Startsignal „Weiter im Mix“ | `Robot_run_cont` | Bool | |
| Fertigmeldung „Presse beladen“ | `Robot_done_load` | Bool | |
| Fertigmeldung „Presse entladen“ | `Robot_done_unload` | Bool | |
| Fertigmeldung „Roboter hat Band beladen“ | `Robot_done_dbelt` | Bool | |

*Tabelle 21 - Datenebene 3: Variablen zur Kommunikation mit dem Roboter*

```
%%44 -BOOL -VAR_GLOBAL
"Press_rdy_take" : "die Bereitschaftsmeldung 'Presse ist für
Beladung bereit'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%45 -BOOL -VAR_GLOBAL
"Robot_run_load" : "das Startsignal 'Roboter belädt Presse'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%46 -BOOL -VAR_GLOBAL
"Robot_done_load" : "die Fertigmeldung 'Roboter hat Teil in
Presse gelegt'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"
```

```
%%47 -BOOL -VAR_GLOBAL
"Press_rdy_give" : "die Bereitschaftsmeldung 'Presse ist zur
Übergabe eines Teils bereit'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%48 -BOOL -VAR_GLOBAL
"Dbelt_rdy_take" : "die Bereitschaftsmeldung 'Ablageband ist
zur Übernahme eines Teils bereit'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%49 -BOOL -VAR_GLOBAL
"Robot_run_unload" : "das Startsignal 'Roboter entlädt
Presse'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%50 -BOOL -VAR_GLOBAL
"Robot_done_unload" : "die Fertigmeldung 'Roboter hat Teil aus
Presse genommen'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%51 -BOOL -VAR_GLOBAL
"Robot_done_dbelt" : "die Fertigmeldung 'Roboter hat Teil auf
Ablageband gelegt'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%52 -BOOL -VAR_GLOBAL
"Robot_run_mix" : "das Startsignal 'Roboter be- und entlädt
Presse'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%53 -BOOL -VAR_GLOBAL
```

```
"Robot_wait_mix" : "die Bereitschaftsmeldung 'Roboter wartet
vor Presse'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%54 -BOOL -VAR_GLOBAL
"Robot_run_cont" : "das Startsignal 'Roboter weiter im Mix'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"
```

*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 39<<< (einfache Forderung - direkt) */*

*Wenn "die Produktionszelle 1" "im Automatikbetrieb" ist und "die Bereitschaftsmeldung 'Hubdrehtisch ist zur Übergabe eines Teils bereit" ist "gesetzt" und "die Bereitschaftsmeldung 'Presse ist zur Übergabe eines Teils bereit'" ist "gesetzt" und "die Bereitschaftsmeldung 'Ablageband ist zur Übernahme eines Teils bereit'" ist "gesetzt" , dann muss "die Bereitschaftsmeldung 'Hubdrehtisch ist zur Übergabe eines Teils bereit" unmittelbar "zurückgesetzt" werden und "die Bereitschaftsmeldung 'Presse ist zur Übergabe eines Teils bereit'" muss unmittelbar "zurückgesetzt" werden und "die Bereitschaftsmeldung 'Ablageband ist zur Übernahme eines Teils bereit'" muss unmittelbar "zurückgesetzt" werden und "das Startsignal 'Roboter be- und entlädt Presse'" muss unmittelbar "gesetzt" werden .*

```
AG ( (rdy_in & cell_1_state=1 & Table_rdy_give &
Press_rdy_give & Dbelt_rdy_take) -> A[!rdy_plc U (rdy_plc
& !Table_rdy_give & !Press_rdy_give & !Dbelt_rdy_take &
Robot_run_mix) ] )
```

*/* >>>Satz 40<<< (einfache Forderung - direkt) */*

*Wenn "die Produktionszelle 1" "im Automatikbetrieb" ist und "die Bereitschaftsmeldung 'Roboter wartet vor Presse'" ist "gesetzt" und "die Bereitschaftsmeldung 'Presse ist für Beladung bereit'" ist "gesetzt" , dann muss "die Bereitschaftsmeldung 'Roboter wartet vor Presse'" unmittelbar "zurückgesetzt" werden und "die Bereitschaftsmeldung 'Presse ist für Beladung bereit'" muss unmittelbar "zurückgesetzt" werden und "das Startsignal 'Roboter weiter im Mix''" muss unmittelbar "gesetzt" werden .*

```
AG ( (rdy_in & cell_1_state=1 & Robot_wait_mix &
Press_rdy_take) -> A[!rdy_plc U (rdy_plc & !Robot_wait_mix
& !Press_rdy_take & Robot_run_cont) ] )
```

*/* >>>Satz 41<<< (einfache Forderung - direkt) */*

*Wenn "die Produktionszelle 1" "im Automatikbetrieb" ist und "die Bereitschaftsmeldung 'Hubdrehtisch ist zur Übergabe eines Teils bereit" ist "gesetzt" und "die Bereitschaftsmeldung 'Presse ist für Beladung bereit'" ist "gesetzt" , dann muss "die Bereitschaftsmeldung 'Hubdrehtisch ist zur Übergabe eines Teils bereit" unmittelbar "zurückgesetzt" werden und "die Bereitschaftsmeldung 'Presse ist für Beladung bereit'" muss unmittelbar "zurückgesetzt" werden und "das Startsignal 'Roboter belädt Presse'" muss unmittelbar "gesetzt" werden .*

```
AG ( (rdy_in & cell_1_state=1 & Table_rdy_give &
Press_rdy_take) -> A[!rdy_plc U (rdy_plc & !Table_rdy_give
& !Press_rdy_take & Robot_run_load) ] )
```

*/* >>>Satz 42<<< (einfache Forderung - direkt) */*

*Wenn "die Produktionszelle 1" "im Automatikbetrieb" ist und "die Bereitschaftsmeldung 'Presse ist zur Übergabe eines Teils bereit'" ist "gesetzt" und "die Bereitschaftsmeldung 'Ablageband ist zur Übernahme eines Teils bereit'" ist "gesetzt" und "die Bereitschaftsmeldung 'Hubdrehtisch ist zur Übergabe eines Teils bereit" ist "nicht gesetzt" , dann muss "die Bereitschaftsmeldung 'Presse ist zur Übergabe eines Teils bereit'" unmittelbar "zurückgesetzt" werden und "die Bereitschaftsmeldung 'Ablageband ist zur Übernahme eines Teils bereit'" muss unmittelbar "zurückgesetzt" werden und "das Startsignal 'Roboter entlädt Presse'" muss unmittelbar "gesetzt" werden .*

```
AG ( (rdy_in & cell_1_state=1 & Press_rdy_give &
Dbelt_rdy_take & !Table_rdy_give) -> A[!rdy_plc U (rdy_plc
& !Press_rdy_give & !Dbelt_rdy_take & Robot_run_unload) ]
)
```

*Informelle Spezifikation weiterer Eigenschaften:*

Es darf keine Beladung erfolgen, wenn die Presse nicht zur Beladung bereit ist.

Es darf kein Entladen erfolgen, wenn die Presse nicht zur Übergabe bereit ist.

Es darf keine Beladung erfolgen, wenn der Hubdrehtisch nicht zur Übergabe bereit ist.

Es darf kein Entladen erfolgen, wenn das Ablageband nicht zur Übernahme bereit ist.

Der Mixbetrieb darf nicht gestartet werden, wenn der Hubdrehtisch nicht zur Übergabe bereit ist.

Der Mixbetrieb darf nicht gestartet werden, wenn die Presse nicht zur Übergabe bereit ist.

Der Mixbetrieb darf nicht gestartet werden, wenn das Ablageband nicht zur Übername bereit ist.

Der Mixbetrieb darf nicht fortgesetzt werden, wenn die Presse nicht für eine Beladung bereit ist.

*Formale Spezifikation mit dem SFS-Editor*

*/* >>>Satz 43<<< (erweiterte Möglichkeit - Zustand) */*

*"das Startsignal 'Roboter belädt Presse'" darf nur "gesetzt" sein , wenn gleichzeitig "die Bereitschaftsmeldung 'Presse ist für Beladung bereit'" "gesetzt" ist .*

```
AG !( rdy_plc & !(Press_ready_take) & (Robot_run_load) )
```

*/* >>>Satz 44<<< (erweiterte Möglichkeit - Zustand) */*

*"das Startsignal 'Roboter entlädt Presse'" darf nur "gesetzt" sein , wenn gleichzeitig "die Bereitschaftsmeldung 'Presse ist zur Übergabe eines Teils bereit'" "gesetzt" ist .*

```
AG !( rdy_plc & !(Press_ready_give) & (Robot_run_unload) )
```

*/* >>>Satz 45<<< (erweiterte Möglichkeit - Zustand) */*

*"das Startsignal 'Roboter belädt Presse'" darf nur "gesetzt" sein , wenn gleichzeitig "die Bereitschaftsmeldung 'Hubdrehtisch ist zur Übergabe eines Teils bereit" "gesetzt" ist .*

```
AG !( rdy_plc & !(Table_ready_give) & (Robot_run_load) )
```

*/* >>>Satz 46<<< (erweiterte Möglichkeit - Zustand) */*

*"das Startsignal 'Roboter entlädt Presse'" darf nur "gesetzt" sein , wenn gleichzeitig "die Bereitschaftsmeldung 'Ablageband ist zur Übernahme eines Teils bereit'" "gesetzt" ist .*

```
AG !( rdy_plc & !(Dbelt_ready_take) & (Robot_run_unload) )
```

*/* >>>Satz 47<<< (erweiterte Möglichkeit - Zustand) */*

*"das Startsignal 'Roboter be- und entlädt Presse'" darf nur "gesetzt" sein , wenn gleichzeitig "die Bereitschaftsmeldung 'Hubdrehtisch ist zur Übergabe eines Teils bereit" "gesetzt" ist und "die Bereitschaftsmeldung 'Presse ist zur Übergabe eines Teils bereit'" ist "gesetzt" und "die Bereitschaftsmeldung 'Ablageband ist zur Übernahme eines Teils bereit'" ist "gesetzt" .*

```
AG !( rdy_plc & !(Table_ready_give & Press_ready_give &
Dbelt_ready_take) & (Robot_run_mix) )
```

*/* >>>Satz 48<<< (erweiterte Möglichkeit - Zustand) */*

*"das Startsignal 'Roboter weiter im Mix'" darf nur "gesetzt" sein , wenn gleichzeitig "die Bereitschaftsmeldung 'Presse ist für Beladung bereit'" "gesetzt" ist .*

```
AG !( rdy_plc & !(Press_ready_take) & (Robot_run_cont) )
```

#### *A.7.2.6 Kommunikation mit der Presse*

Die Presse ist die eigentliche Bearbeitungsmaschine innerhalb der Produktionszelle 1. Sie wird durch den Roboter mit Rohteilen beladen (Roboterarm 1) und auch wieder durch den Roboter entladen (Roboterarm 2).

Die Presse wird durch den Roboter beladen und auch wieder entladen. Es existiert wiederum kein Sensor, um die Beladung der Presse direkt feststellen zu können. Durch die Presse sind zwei verschiedene Bewegungsphasen auszuführen: nachdem der Roboter die Presse mit einem Rohteil beladen hat, vollzieht diese die Arbeitsbewegung und bearbeitet das Rohteil. Danach bewegt sich die Presse in eine Position, in der der Roboter die Presse entladen kann. Erst danach bewegt sich die Presse wieder in die Beladeposition.

Die Bewegung der Presse wird in zwei Phasen unterteilt:

1. der eigentliche Pressvorgang,
2. die Vorbereitung auf die Aufnahme eines neuen Teils.

zu 1. Es wird eine Meldung des Roboters benötigt, dass er ein neues Rohteil in die Presse gelegt hat. Danach kann die Presse mit dem Pressvorgang beginnen.

zu 2. Es wird eine Meldung des Roboters benötigt, dass er das Fertigteil aus der Presse entnommen hat. Danach kann die Presse in die Ausgangsstellung zurückkehren.

*Definition neuer Daten mit dem SFS-Editor*

Variablen zur Kommunikation mit der Presse

| **Technische Bezeichnung** | **Variable** | **Datentyp** | **Kommentar** |
|---|---|---|---|
| Startsignal „Pressen“ | `Press_run_work` | Bool | |
| Startsignal „Beladen Vorbereiten“ | `Press_run_prepare` | Bool | |

*Tabelle 22 - Datenebene 3: Variablen zur Kommunikation mit der Presse*

```
%%55 -BOOL -VAR_GLOBAL
"Press_run_work" : "das Startsignal 'Presse soll pressen'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%56 -BOOL -VAR_GLOBAL
"Press_run_prepare" : "das Startsignal 'Presse soll Beladung
vorbereiten'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
```

```
FALSE_O : "zurückgesetzt"
```

*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 52<<< (einfache Forderung - direkt) */*

*Wenn "die Produktionszelle 1" "im Automatikbetrieb" ist und "die Fertigmeldung 'Roboter hat Teil in Presse gelegt'" ist "gesetzt" , dann muss "die Fertigmeldung 'Roboter hat Teil in Presse gelegt'" unmittelbar "zurückgesetzt" werden und "das Startsignal 'Presse soll pressen'" muss unmittelbar "gesetzt" werden .*

```
AG ( (rdy_in & cell_1_state=1 & Robot_done_load) ->
A[!rdy_plc U (rdy_plc & !Robot_done_load & Press_run_work)
] )
```

*/* >>>Satz 53<<< (einfache Forderung - direkt) */*

*Wenn "die Produktionszelle 1" "im Automatikbetrieb" ist und "die Fertigmeldung 'Roboter hat Teil aus Presse genommen'" ist "gesetzt" , dann muss "die Fertigmeldung 'Roboter hat Teil aus Presse genommen'" unmittelbar "zurückgesetzt" werden und "das Startsignal 'Presse soll Beladung vorbereiten'" muss unmittelbar "gesetzt" werden .*

```
AG ( (rdy_in & cell_1_state=1 & Robot_done_unload) ->
A[!rdy_plc U (rdy_plc & !Robot_done_unload &
Press_run_prepare) ] )
```

*Informelle Spezifikation weiterer Eigenschaften:*

Die Presse darf nicht gestartet werden, wenn der Roboter kein Teil in die Presse gelegt hat.

Die Presse darf nicht vorbereitet werden, wenn der Roboter das Teil nicht aus der Presse genommen hat.

*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 54<<< (einfaches Verbot - selbstbegrenzt) */*

*"das Startsignal 'Presse soll pressen'" darf nicht irgendwann "gesetzt" werden , solange "die Fertigmeldung 'Roboter hat Teil in Presse gelegt'" "nicht gesetzt" ist .*

```
AG !( rdy_plc & !Robot_done_load & (Press_run_work) )
```

*/* >>>Satz 55<<< (einfaches Verbot - selbstbegrenzt) */*

*"das Startsignal 'Presse soll Beladung vorbereiten'" darf niemals "gesetzt" werden , solange "die Fertigmeldung 'Roboter hat Teil aus Presse genommen'" "nicht gesetzt" ist .*

```
AG !( rdy_plc & !Robot_done_unload & (Press_run_prepare) )
```

### *A.7.2.7 Kommunikation mit dem Ablageband*

Das Ablageband dient dazu, die Fertigteile, die der Roboter auf des Ablageband abgelegt hat, zum Kran zu befördern.

Im Gegensatz zum Zuführband besteht der Transportvorgang auf dem Ablageband nur aus einer Phase, für die Entnahme des Teils vom Anlageband ist nur der Kran zuständig. Der Transport eines Teils ist also erst dann möglich, wenn der Kran das vorherige Teil vom Ablageband genommen hat. Außerdem wird für die Steuerung des Ablagebands ein Signal benötigt, wann der Roboter ein Teil auf dem Ablageband abgelegt hat. Erst dann kann der Transport beginnen. Aus Effizienzgründen kann jedoch eine Beladung des Ablagebands durch den Roboter auch bereits dann erfolgen, wenn ein Teil am Bandende liegt, der Kran es jedoch noch nicht entnommen hat. Mit dem Transport wird dann erst begonnen, wenn der Kran das Teil entnommen hat.

*Definition neuer Daten mit dem SFS-Editor*

Es wird eine Meldung über die Beladung des Ablagebands durch den Roboter benötigt. Es muss weiterhin eine Meldung darüber geben, wann der Kran ein Teil vom Ablageband genommen hat. Es wird ein Startsignal zur Ausführung des Transports definiert.

Variablen zur Kommunikation mit dem Ablageband

| **Technische Bezeichnung** | **Variable** | **Datentyp** | **Kommentar** |
|---|---|---|---|
| Startsignal „Ablageband transportiert Teil zum Kran“ | Dbelt_run | Bool | |
| Meldung „Ablageband für Übergabe eines Fertigteils bereit“ | Dbelt_rdy_give | Bool | |

*Tabelle 23 - Datenebene 3: Variablen zur Kommunikation mit dem Ablageband*

```
%%57 -BOOL -VAR_GLOBAL
"Dbelt_run" : "das Startsignal 'Ablageband soll Teil
transportieren'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%58 -BOOL -VAR_GLOBAL
"Dbelt_rdy_give" : "die Bereitschaftsmeldung 'Ablageband ist
zur Übergabe eines Teils bereit'"
TRUE_I : "gesetzt"
```

```
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"
```

*Formale Spezifikation mit dem SFS-Editor:*

Ablageband

*/* >>>Satz 56<<< (einfache Forderung - direkt) */*

*Wenn "die Produktionszelle 1" "im Automatikbetrieb" ist und "die Fertigmeldung 'Roboter hat Teil auf Ablageband gelegt'" ist "gesetzt" und "die Fertigmeldung 'Kran hat Fertigteil vom Ablageband genommen'" ist "gesetzt" , dann muss "die Fertigmeldung 'Roboter hat Teil auf Ablageband gelegt'" unmittelbar "zurückgesetzt" werden und "die Fertigmeldung 'Kran hat Fertigteil vom Ablageband genommen'" muss unmittelbar "zurückgesetzt" werden und "das Startsignal 'Ablageband soll Teil transportieren'" muss unmittelbar "gesetzt" werden .*

```
AG ( (rdy_in & cell_1_state=1 & Robot_done_dbelt &
Crane_done_take_dbelt) -> A[!rdy_plc U (rdy_plc &
!Robot_done_dbelt & !Crane_done_take_dbelt & Dbelt_run) ]
)
```

*Informelle Spezifikation weiterer Eigenschaften:*

Das Ablageband darf nicht starten, solange der Roboter kein Teil auf das Ablageband gelegt hat.

Das Ablageband darf nicht starten, solange der Kran das Teil nicht vom Band genommen hat.

*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 57<<< (einfaches Verbot - Zustand) */*

*"das Startsignal 'Ablageband soll Teil transportieren'" darf nicht gleichzeitig "gesetzt" sein , wenn "die Fertigmeldung 'Roboter hat Teil auf Ablageband gelegt'" "nicht gesetzt" ist .*

```
AG !( rdy_plc & !Robot_done_dbelt & (Dbelt_run) )
```

*/* >>>Satz 58<<< (einfaches Verbot - Zustand) */*

*"das Startsignal 'Ablageband soll Teil transportieren'" darf nicht gleichzeitig "gesetzt" sein , wenn "die Fertigmeldung 'Kran hat Fertigteil vom Ablageband genommen'" "nicht gesetzt" ist .*

```
AG !( rdy_plc & !Crane_done_take_dbelt & (Dbelt_run) )
```

### *A.7.2.8 Kommunikation mit dem Kran (II)*

Die Steuerung der Produktionszelle muss an die Steuerung des Krans ein Signal geben, wenn dieser ein Fertigteil vom Ablageband holen soll. Dazu müssen der Zellensteuerung die folgenden Informationen bekannt sein:

- auf dem Ablageband liegt ein Fertigteil bereit,
- der Ausgangspuffer ist frei.

Für den Ausgangspuffer war vorausgesetzt, dass dieser jederzeit Teile aufnehmen kann.

*Definition neuer Daten mit dem SFS-Editor*

Das Ablageband muss eine Meldung generieren, wenn es zur Übergabe eines Fertigteils bereit ist (existiert bereits). Es wird ein Startsignal definiert, das den Kran zum Holen des Fertigteils veranlasst.

Variablen zur Kommunikation mit dem Kran

| **Technische Bezeichnung** | **Variable** | **Datentyp** | **Kommentar** |
|---|---|---|---|
| Startsignal „Fertigteil holen“ | `Crane_run_dbelt` | Bool | |
| Fertigmeldung „Kran hat Teil vom Ablageband genommen“ | `Crane_done_take_dbelt` | Bool | |

*Tabelle 24 - Datenebene 3: Variablen zur Kommunikation mit dem Kran*

```
%%59 -BOOL -VAR_GLOBAL
"Crane_run_dbelt" : "das Startsignal 'Kran holt Fertigteil von
Ablageband'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"

%%60 -BOOL -VAR_GLOBAL
"Crane_done_take_dbelt" : "die Fertigmeldung 'Kran hat
Fertigteil vom Ablageband genommen'"
TRUE_I : "gesetzt"
FALSE_I : "nicht gesetzt"
TRUE_O : "gesetzt"
FALSE_O : "zurückgesetzt"
```

*Formale Spezifikation mit dem SFS-Editor:*

| |
|---|
| */* >>>Satz 60<<< (einfache Forderung - direkt) */* |
| *Wenn "die Produktionszelle 1" "im Automatikbetrieb" ist und "die Bereitschaftsmeldung 'Ablageband ist zur Übergabe eines Teils bereit'" ist "gesetzt" und "der Ausgangspuffer" ist "frei" , dann muss "die Bereitschaftsmeldung 'Ablageband ist zur Übergabe eines Teils bereit'" unmittelbar "zurückgesetzt" werden und muss "das Startsignal 'Kran holt Fertigteil von Ablageband'" unmittelbar "gesetzt" werden .* |

```
AG ( (rdy_in & cell_1_state=1 & Dbelt_rdy_give &
!Piece_at_drain) -> A[!rdy_plc U (rdy_plc &
!Dbelt_rdy_give & Crane_run_dbelt) ] )
```

*Definition weiterer Eigenschaften mit dem SFS-Editor:*

Der Kran darf nicht starten, wenn das Ablageband nicht bereit ist.

Der Kran darf nicht starten, wenn der Ausgangspuffer nicht frei ist.

*Formale Spezifikation mit dem SFS-Editor:*

| |
|---|
| */* >>>Satz 61<<< (einfaches Verbot - Zustand) */* |
| *"das Startsignal 'Kran holt Fertigteil vom Ablageband'" darf nicht gleichzeitig "gesetzt" sein , wenn "die Bereitschaftsmeldung 'Ablageband ist zur Übergabe eines Teils bereit'" "nicht gesetzt" ist .* |

```
AG !( rdy_plc & !Dbelt_ready_give & (Crane_run_dbelt) )
```

| |
|---|
| */* >>>Satz 62<<< (einfaches Verbot - Zustand) */* |
| *"das Startsignal 'Kran holt Fertigteil vom Ablageband'" darf nicht gleichzeitig "gesetzt" sein , wenn "der Ausgangspuffer" "belegt" ist .* |

```
AG !( rdy_plc & Piece_at_drain & (Crane_run_dbelt) )
```

### *A.7.3 Geräteebene 2 - Steuerung des Transportsystems*

Da sich die Darstellung der Fallstudie auf die Spezifikation der Produktionszelle konzentriert, wird die Spezifikation des Transportsystems nur ansatzweise dargestellt. Diese erfolgt jedoch nach den gleichen Prinzipien, wie sie bereits im Falle der Produktionszelle dargestellt und angewandt wurden.

Das Transportsystem dient innerhalb der Produktionsanlage dazu, die einzelnen Produktionszellen mit Rohteilen zu versorgen bzw. die bearbeiteten Fertigteile wieder von den Produktionszellen abzuholen.

*Gegebene Daten:*

Das Transportsystem erhält von der Auftragsverwaltung neue Transportaufträge und informiert die Auftragsverwaltung über seinen Bereitschaftszustand und den Status der Ausführung des vorherigen Auftrags.

*Details des Transportauftrags: siehe Tabelle 10 (Anhang, Seite 142)*

*Informelle Spezifikation:*

Das Transportsystem meldet an die Auftragsverwaltung, wenn es zur Ausführung eines neuen Transportauftrags bereit ist. Sobald die Auftragsverwaltung einen neuen Transportauftrag generiert hat (die Variable „`tr_kind`“ verändert ihren Wert von „0“ auf „1“ oder „2“) beginnt das Transportsystem automatisch mit der Ausführung dieses Auftrags. Dazu muss das Transportsystem selbständig bestimmte Positionen innerhalb der Gesamtanlage anfahren können und die Übergabe bzw. die Übernahme von Werkstücken realisieren:

- Wenn sich das (leere) Transportsystem am Rohteilelager befindet, übernimmt es selbständig die festgelegte Anzahl von Rohteilen.
- Nachdem das (beladene) Transportsystem die Produktionszelle erreicht hat, übergibt es selbständig alle Rohteile an den Eingangspuffer der Produktionszelle.
- Nachdem das (leere) Transportsystem die Produktionszelle erreicht hat, übernimmt es selbständig alle Fertigteile aus dem Ausgangspuffer der Produktionszelle
- Nachdem das (beladene) Transportsystem das Fertigteillager erreicht hat, übergibt es selbständig alle Fertigteile an das Fertigteillager.

Nach der Beendigung des Transportauftrags generiert das Transportsystem seine Fertig- bzw. Bereitschaftsmeldung.

*Definition neuer Daten mit dem SFS-Editor*

Es müssen Daten zur Lagebestimmung definiert werden, zusätzlich dazu werden Signale vereinbart, die die Einnahme von bestimmten Positionen anzeigen.

*Auf die Details dieser Daten wird in dieser Arbeit nicht eingegangen.*

Zur Realisierung der Materialübernahme aus dem Rohteilelager muss eine Zählung der übernommenen Rohteile erfolgen. Die Übernahme ist beendet, wenn die gewünschte Anzahl von Rohteilen entnommen wurde. Dasselbe gilt für die Übernahme der Fertigteile aus dem Ausgangspuffer der Produktionszelle. Die Übergabe der Rohteile an die Produktionszelle bzw. die Übergabe der Fertigteile an das Fertigteillager erfolgt solange, bis das Transportsystem leer ist.

*Auf die Details dieser Daten wird in dieser Arbeit ebenfalls nicht eingegangen.*

*Formale Spezifikation mit dem SFS-Editor:*

Auf die Details der formalen Spezifikation wird in dieser Arbeit nicht eingegangen. Der Leser möge sich jedoch vorstellen, dass die Vorgehensweise analog zu der Spezifikation der Produktionszelle 1 ist.

### *A.7.4 Geräteebene 3 - Steuerung des Krans*

Aufgabe:

Die Steuerung des Krans erhält von der übergeordneten Zellensteuerung Startsignale zur Ausführung bestimmter Aktionen. Durch verschiedene Sensoren und Aktoren ist die Kransteuerung mit den Maschinenelementen verbunden.

*Gegebene Daten*

| **Technische Bezeichnung** | **Variable** | **Datentyp** | **Kommentar** |
|---|---|---|---|
| Startsignal „Rohteil holen“ | `Crane_run_source` | Bool | |
| Fertigmeldung Kran | `Crane_done_give_fbelt` | Bool | Kran hat Rohteil auf Zuführband abgelegt |
| Startsignal „Fertigteil holen“ | `Crane_run_dbelt` | Bool | |
| Fertigmeldung „Kran hat Teil vom Ablageband genommen“ | `Crane_done_take_dbelt` | Bool | |

*Definition neuer Daten mit dem SFS-Editor*

Sensoren des Krans

| **Technische Bezeichnung** | **SPS-Variable** | **Datentyp** | **Kommentar** |
|---|---|---|---|
| Schalter Kranposition Zuführband | `Crane_at_fbelt` | Bool | TRUE - Kran ist am Zuführband |
| Schalter Kranposition Ablageband | `Crane_at_dbelt` | Bool | TRUE - Kran ist am Ablageband |
| Schalter Kranposition Eingangspuffer | `Crane_at_source` | Bool | TRUE - Kran ist am Eingangspuffer |
| Schalter Kranposition Ausgangspuffer | `Crane_at_drain` | Bool | TRUE - Kran ist am Ausgangspuffer |
| Wegmesssystem Greiferhöhe | `Crane_height` | Integer | 0,00 - Greifer ist oben<br>0,35 - Greifer ist auf Ausgangspuffer<br>0,66 - Greifer auf Abl.-band<br>0;85 - Greifer ist auf Eingangspuffer |

| | | | |
|---|---|---|---|
| | | | 0,94 - Greifer auf Zuf.-band<br>1,00 - Greifer ist unten |

*Tabelle 25 - Datenebene 4: Sensoren des Krans*

```
%%61 -BOOL -VAR
"Crane_at_dbelt" : "der Kran"
TRUE_I : "am Ablageband"
FALSE_I : "nicht am Ablageband"

%%62 -BOOL -VAR
"Crane_at_fbelt" : "der Kran"
TRUE_I : "am Zuführband"
FALSE_I : "nicht am Zuführband"

%%63 -BOOL -VAR_INPUT
"Crane_at_source" : "der Kran"
TRUE_I : "am Eingangspuffer"
FALSE_I : "nicht am Eingangspuffer"

%%64 -BOOL -VAR_INPUT
"Crane_at_drain" : "der Kran"
TRUE_I : "am Ausgangspuffer"
FALSE_I : "nicht am Ausgangspuffer"

%%65 -INT -VAR
"Crane_height" : "der Greifer des Krans"
0_I : "oben"
0.35_I : "auf Ausgangspuffer"
0.66_I : "auf dem Ablageband"
0.85_I : "auf Eingangspuffer"
0.94_I : "auf dem Zuführband"
1.0_I : "unten"
```

Aktoren des Krans

| **Technische Bezeichnung** | **SPS-Variable** | **Datentyp** | **Kommentar** |
|---|---|---|---|
| Motor Kranposition (L/R) | `Crane_to_fbelt` | Bool | TRUE - Kran fährt zum Zuf.-band |
| | `Crane_to_dbelt` | Bool | TRUE - Kran fährt zum Abl.-band |
| Motor Greiferhöhe (L/R) | `Crane_lift` | Bool | TRUE - Greifer wird angehoben |
| | `Crane_lower` | Bool | TRUE - Greifer wird abgesenkt |
| Greifer Kran | `Crane_mag_on` | Bool | TRUE - Magnet eingeschaltet |

*Tabelle 26 - Datenebene 4: Aktoren des Krans*

```
%%66 -BOOL -VAR
"Crane_to_fbelt" : "die Bewegung des Krans nach links (zum
Zuführband)"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
TRUE_O : "gestartet"
FALSE_O : "gestoppt"

%%67 -BOOL -VAR
"Crane_to_dbelt" : "die Bewegung des Krans nach rechts
(Ablageband)"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
TRUE_O : "gestartet"
FALSE_O : "gestoppt"

%%68 -BOOL -VAR
"Crane_lift" : "das Anheben des Krangreifers"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
TRUE_O : "gestartet"
FALSE_O : "gestoppt"

%%69 -BOOL -VAR
"Crane_lower" : "das Absenken des Krangreifers"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
TRUE_O : "gestartet"
FALSE_O : "gestoppt"

%%70 -BOOL -VAR
"Crane_mag_on" : "der Magnet des Krangreifers"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
TRUE_O : "aktiviert"
FALSE_O : "deaktiviert"
```

*Informelle Spezifikation:*

Der Steuerung des Krans verbleibt solange in Ruhe, bis durch die Zellensteuerung entweder das Startsignal zum Transport eines Rohteils oder eines Fertigteils gesetzt wird. Danach wird die geforderte Transportroutine bearbeitet.

Rohteile vom Eingangspuffer holen

Damit der Kran den Eingangspuffer korrekt anfahren kann, muss er zunächst seine relative Position zu ihm bestimmen. Erst dann kann entschieden werden, in welche Richtung der Kran fahren soll, d. h. welches Aktorsignal zu aktiveren ist. Prinzipiell

kann davon ausgegangen werden, dass sich der Kran nach einem zuvor erfolgten Transportvorgang entweder am Zuführband oder am Ausgangspuffer befindet. Diese beiden Positionen, und auch wenn sich der Kran bereits am Eingangspuffer oder am Ablageband befindet, können durch die vorhandenen Sensoren erkannt werden. Sollte sich der Kran an einer unbekannten Position befinden (wenn also keiner der vier Sensoren aktiv sein), so muss zunächst eine Fahrtrichtung festgelegt und durch nachfolgende erstmalige Erkennung eines Sensors die Position bestimmt werden. Unter Umständen ist dann die Fahrtrichtung wieder umzukehren. Hat der Kran jedoch die Position des Eingangspuffers erreicht, so wird der Magnetgreifer auf die Höhe des Eingangspuffers abgesenkt. Dann wird der Magnetgreifer aktiviert. Danach beginnt das Anheben des Greifers. Sobald dieser wieder oben ist, fährt der Kran zum Zuführband. Dort angekommen, wird der Greifer wieder abgesenkt. Auf der Höhe des Zuführbands wird der Greifer angehalten und der Magnet wird abgeschalten. Gleichzeitig kann die Meldung „Kran hat Teil auf Zuführband gelegt“ gesetzt werden. Danach wird der Greifer wieder angehoben. Sobald der Greifer wieder in seiner oberen Position ist, wird er gestoppt und gelangt wieder in den Ruhezustand.

Fertigteile zum Ausgangspuffer bringen

Bei diesem Transportvorgang ist die Fahrtrichtung eindeutig bestimmbar, da das Ablageband am äußeren Ende des Verfahrbereichs liegt. Die anschließenden Vorgänge verlaufen analog zu den soeben beschriebenen Vorgängen. Am Ablageband muss der Greifer bis auf die Höhe des Ablagebands abgesenkt werden, der Magnetgreifer wird aktiviert und der Greifer wieder angehoben. Gleichzeitig kann auch die Meldung „Kran hat Teil vom Ablageband genommen“ gesetzt werden. Nach der Fahrt zum Ausgangspuffer wird der Greifer wieder abgesenkt, der Magnet wird deaktiviert und der Greifer wieder angehoben. Nachdem der Greifer seine obere Position wieder erreicht hat, wird er gestoppt und gelangt wieder in den Ruhezustand.

Da die gezeigten Teilvorgänge unbedingt in der festgelegten Reihenfolge abzuarbeiten sind, bietet sich die Entwicklung einer Ablaufstruktur an.

*Definition neuer Daten mit dem SFS-Editor:*

Es wird eine Zustandsvariable definiert, die den aktuellen Zustand darstellt, in dem sich der Kran bzw. die Ablaufsteuerung des Krans befindet:

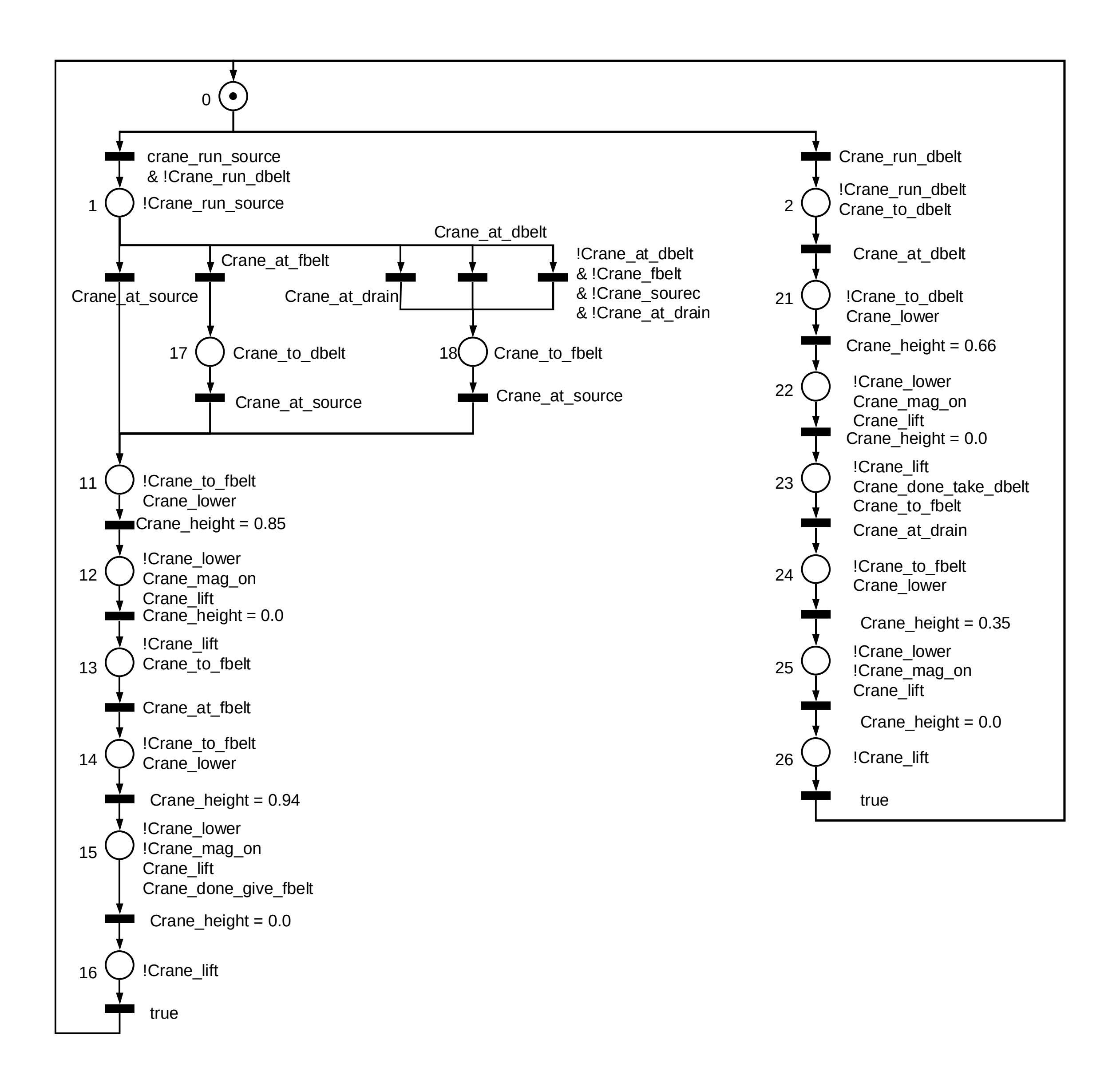
Interne Variable der Steuerung des Krans

| **Technische Bezeichnung** | **SPS-Variable** | **Datentyp** | **Kommentar** |
|---|---|---|---|
| Zustandsvariable des Krans | `Crane_state` | Integer | |

*Tabelle 27 - Datenebene 4: Interne Variable der Steuerung des Krans*

```
%%71 -INT -VAR
"Crane_state" : "die Zustandsvariable des Krans"
0_I : "0"
1_I : "1"
2_I : "2"
11_I : "11"
12_I : "12"
```

```
13_I : "13"
14_I : "14"
15_I : "15"
16_I : "16"
17_I : "17"
18_I : "18"
21_I : "21"
22_I : "22"
23_I : "23"
24_I : "24"
25_I : "25"
26_I : "26"
0_O : "auf 0 gesetzt"
1_O : "auf 1 gesetzt"
2_O : "auf 2 gesetzt"
11_O : "auf 11 gesetzt"
12_O : "auf 12 gesetzt"
13_O : "auf 13 gesetzt"
14_O : "auf 14 gesetzt"
15_O : "auf 15 gesetzt"
16_O : "auf 16 gesetzt"
17_O : "auf 17 gesetzt"
18_O : "auf 18 gesetzt"
21_O : "auf 21 gesetzt"
22_O : "auf 22 gesetzt"
23_O : "auf 23 gesetzt"
24_O : "auf 24 gesetzt"
25_O : "auf 25 gesetzt"
26_O : "auf 26 gesetzt"
```




*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 64<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Krans" "0" ist und "das Startsignal 'Kran holt Rohteil vom Eingangspuffer'" ist "gesetzt" und "das Startsignal 'Kran holt Fertigteil von Ablageband'" ist "nicht gesetzt" , dann muss "das Startsignal 'Kran holt Rohteil vom Eingangspuffer'" unmittelbar "zurückgesetzt" werden und "die Zustandsvariable des Krans" muss unmittelbar "1" werden .*

```
AG ( (rdy_in & Crane_state=0 & Crane_run_source &
!Crane_run_dbelt) -> A[!rdy_plc U (rdy_plc &
!Crane_run_source & Crane_state=1) ] )
```

*/* >>>Satz 65<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Krans" "0" ist und "das Startsignal 'Kran holt Fertigteil von Ablageband'" ist "gesetzt" , dann muss "das Startsignal 'Kran holt*

*Fertigteil von Ablageband'" unmittelbar "zurückgesetzt" werden und "die Zustandsvariable des Krans" muss unmittelbar "2" werden und "die Bewegung des Krans nach rechts (Ablageband)" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Crane_state=0 & Crane_run_dbelt) ->
A[!rdy_plc U (rdy_plc & !Crane_run_dbelt & Crane_state=2 &
Crane_to_dbelt) ] )
```

*/* >>>Satz 66<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Krans" "1" ist und "der Kran" ist "am Eingangspuffer" , dann muss "die Zustandsvariable des Krans" unmittelbar "11" werden und "das Absenken des Krangreifers" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Crane_state=1 & Crane_at_source) ->
A[!rdy_plc U (rdy_plc & Crane_state=11 & Crane_lower) ] )
```

*/* >>>Satz 67<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Krans" "1" ist und "der Kran" ist "am Zuführband" , dann muss "die Zustandsvariable des Krans" unmittelbar "1a" werden und "die Bewegung des Krans nach rechts (Ablageband)" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Crane_state=1 & Crane_at_fbelt) ->
A[!rdy_plc U (rdy_plc & Crane_state=16 & Crane_to_dbelt) ]
)
```

*/* >>>Satz 68<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Krans" "1" ist und "der Kran" ist "am Ausgangspuffer" , dann muss "die Zustandsvariable des Krans" unmittelbar "1b" werden und "die Bewegung des Krans nach links (zum Zuführband)" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Crane_state=1 & Crane_at_drain) ->
A[!rdy_plc U (rdy_plc & Crane_state=17 & Crane_to_fbelt) ]
)
```

*/* >>>Satz 69<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Krans" "1" ist und "der Kran" ist "am Ablageband" , dann muss "die Zustandsvariable des Krans" unmittelbar "1b" werden und "die Bewegung des Krans nach links (zum Zuführband)" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Crane_state=1 & Crane_at_dbelt) ->
A[!rdy_plc U (rdy_plc & Crane_state=17 & Crane_to_fbelt) ]
```

```
)
```

*/* >>>Satz 70<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Krans" "1" ist und "der Kran" ist "nicht am Ablageband" und "der Kran" ist "nicht am Zuführband" und "der Kran" ist "nicht am Eingangspuffer" und "der Kran" ist "nicht am Ausgangspuffer" , dann muss "die Zustandsvariable des Krans" unmittelbar "1b" werden und "die Bewegung des Krans nach links (zum Zuführband)" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Crane_state=1 & !Crane_at_dbelt &
!Crane_at_fbelt & !Crane_at_source & !Crane_at_drain) ->
A[!rdy_plc U (rdy_plc & Crane_state=17 & Crane_to_fbelt) ]
)
```

*/* >>>Satz 71<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Krans" "1a" ist und "der Kran" ist "am Eingangspuffer" , dann muss "die Bewegung des Krans nach rechts (Ablageband)" unmittelbar "gestoppt" werden und "die Zustandsvariable des Krans" muss unmittelbar "11" werden und "das Absenken des Krangreifers" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Crane_state=16 & Crane_at_source) ->
A[!rdy_plc U (rdy_plc & !Crane_to_dbelt & Crane_state=11 &
Crane_lower) ] )
```

*/* >>>Satz 72<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Krans" "1b" ist und "der Kran" ist "am Eingangspuffer" , dann muss "die Bewegung des Krans nach links (zum Zuführband)" unmittelbar "gestoppt" werden und "die Zustandsvariable des Krans" muss unmittelbar "11" werden und "das Absenken des Krangreifers" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Crane_state=17 & Crane_at_source) ->
A[!rdy_plc U (rdy_plc & !Crane_to_fbelt & Crane_state=11 &
Crane_lower) ] )
```

*/* >>>Satz 73<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Krans" "1b" ist und "der Kran" ist "am Zuführband" , dann muss "die Bewegung des Krans nach links (zum Zuführband)" unmittelbar "gestoppt" werden und "die Zustandsvariable des Krans" muss unmittelbar "1c" werden und "die Bewegung des Krans nach rechts (Ablageband)" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Crane_state=17 & Crane_at_fbelt) ->
A[!rdy_plc U (rdy_plc & !Crane_to_fbelt & Crane_state=18 &
```

```
Crane_to_dbelt) ] )
```

*/* >>>Satz 74<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Krans" "1b" ist und "der Kran" ist "am Eingangspuffer" , dann muss "die Bewegung des Krans nach rechts (Ablageband)" unmittelbar "gestoppt" werden und "die Zustandsvariable des Krans" muss unmittelbar "11" werden und "das Absenken des Krangreifers" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Crane_state=17 & Crane_at_source) ->
A[!rdy_plc U (rdy_plc & !Crane_to_dbelt & Crane_state=11 &
Crane_lower) ] )
```

*/* >>>Satz 75<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Krans" "11" ist und "der Greifer des Krans" ist "auf dem Eingangspuffer" , dann muss "das Absenken des Krangreifers" unmittelbar "gestoppt" werden und "die Zustandsvariable des Krans" muss unmittelbar "12" werden und "der Magnet des Krangreifers" muss unmittelbar "aktiviert" werden und "das Anheben des Krangreifers" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Crane_state=11 & Crane_height=0.85) ->
A[!rdy_plc U (rdy_plc & !Crane_lower & Crane_state=12 &
Crane_mag_on & Crane_lift) ] )
```

*/* >>>Satz 76<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Krans" "12" ist und "der Greifer des Krans" ist "oben" , dann muss "das Anheben des Krangreifers" unmittelbar "gestoppt" werden und "die Fertigmeldung 'Kran hat eine Rohteil auf das Zuführband gelegt'" muss unmittelbar "gesetzt" werden und "die Zustandsvariable des Krans" muss unmittelbar "13" werden und "die Bewegung des Krans nach links (zum Zuführband)" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Crane_state=12 & Crane_height=0) ->
A[!rdy_plc U (rdy_plc & !Crane_lift &
Crane_done_give_fbelt & Crane_state=13 & Crane_to_fbelt) ]
)
```

*/* >>>Satz 77<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Krans" "13" ist und "der Kran" ist "am Zuführband" , dann muss "die Bewegung des Krans nach links (zum Zuführband)" unmittelbar "gestoppt" werden und "die Zustandsvariable des Krans" muss unmittelbar "14" werden und "das Absenken des Krangreifers" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Crane_state=13 & Crane_at_fbelt) ->
A[!rdy_plc U (rdy_plc & !Crane_to_fbelt & Crane_state=14 &
```

```
Crane_lower) ] )
```

*/* >>>Satz 78<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Krans" "14" ist und "der Greifer des Krans" ist "auf dem Zuführband" , dann muss "das Absenken des Krangreifers" unmittelbar "gestoppt" werden und "die Zustandsvariable des Krans" muss unmittelbar "15" werden und "der Magnet des Krangreifers" muss unmittelbar "deaktiviert" werden und "das Anheben des Krangreifers" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Crane_state=14 & Crane_height=0.94) ->
A[!rdy_plc U (rdy_plc & !Crane_lower & Crane_state=15 &
!Crane_mag_on & Crane_lift) ] )
```

*/* >>>Satz 79<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Krans" "15" ist und "der Greifer des Krans" ist "oben" , dann muss "das Anheben des Krangreifers" unmittelbar "gestoppt" werden und "die Zustandsvariable des Krans" muss unmittelbar "0" werden .*

```
AG ( (rdy_in & Crane_state=15 & Crane_height=0) ->
A[!rdy_plc U (rdy_plc & !Crane_lift & Crane_state=0) ] )
```

*/* >>>Satz 80<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Krans" "2" ist und "der Kran" ist "am Ablageband" , dann muss "die Bewegung des Krans nach rechts (Ablageband)" unmittelbar "gestoppt" werden und "die Zustandsvariable des Krans" muss unmittelbar "21" werden und "das Absenken des Krangreifers" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Crane_state=2 & Crane_at_dbelt) ->
A[!rdy_plc U (rdy_plc & !Crane_to_dbelt & Crane_state=21 &
Crane_lower) ] )
```

*/* >>>Satz 81<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Krans" "21" ist und "der Greifer des Krans" ist "auf dem Ablageband" , dann muss "das Absenken des Krangreifers" unmittelbar "gestoppt" werden und "die Zustandsvariable des Krans" muss unmittelbar "22" werden und "der Magnet des Krangreifers" muss unmittelbar "aktiviert" werden und "das Anheben des Krangreifers" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Crane_state=21 & Crane_height=0.66) ->
A[!rdy_plc U (rdy_plc & !Crane_lower & Crane_state=22 &
Crane_mag_on & Crane_lift) ] )
```

*/* >>>Satz 82<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Krans" "22" ist und "der Greifer des Krans" ist "oben" , dann muss "das Anheben des Krangreifers" unmittelbar "gestoppt" werden*

*und "die Zustandsvariable des Krans" muss unmittelbar "23" werden und "die Fertigmeldung 'Kran hat Fertigteil vom Ablageband genommen'" muss unmittelbar "gesetzt" werden und "die Bewegung des Krans nach links (zum Zuführband)" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Crane_state=22 & Crane_height=0) ->
A[!rdy_plc U (rdy_plc & !Crane_lift & Crane_state=23 &
Crane_done_take_dbelt & Crane_to_fbelt) ] )
```

*/* >>>Satz 83<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Krans" "23" ist und "der Kran" ist "am Ausgangspuffer" , dann muss "die Bewegung des Krans nach links (zum Zuführband)" unmittelbar "gestoppt" werden und "die Zustandsvariable des Krans" muss unmittelbar "24" werden und "das Absenken des Krangreifers" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Crane_state=23 & Crane_at_drain) ->
A[!rdy_plc U (rdy_plc & !Crane_to_fbelt & Crane_state=24 &
Crane_lower) ] )
```

*/* >>>Satz 84<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Krans" "24" ist und "der Greifer des Krans" ist "auf dem Ausgangspuffer" , dann muss "das Absenken des Krangreifers" unmittelbar "gestoppt" werden und "die Zustandsvariable des Krans" muss unmittelbar "25" werden und "der Magnet des Krangreifers" muss unmittelbar "deaktiviert" werden und "das Anheben des Krangreifers" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Crane_state=24 & Crane_height=0.35) ->
A[!rdy_plc U (rdy_plc & !Crane_lower & Crane_state=25 &
!Crane_mag_on & Crane_lift) ] )
```

*/* >>>Satz 85<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Krans" "25" ist und "der Greifer des Krans" ist "oben" , dann muss "das Anheben des Krangreifers" unmittelbar "gestoppt" werden und "die Zustandsvariable des Krans" muss unmittelbar "0" werden .*

```
AG ( (rdy_in & Crane_state=25 & Crane_height=0) ->
A[!rdy_plc U (rdy_plc & !Crane_lift & Crane_state=0) ] )
```

*Informelle Spezifikation weiterer Eigenschaften (Sicherheit Kran):*

Neben den Funktionsanforderungen müssen durch die Kransteuerung weitere Sicherheitsanforderungen realisiert werden. Hierbei können die folgenden sicherheitskritischen Punkte identifiziert werden:

- Sätze 86, 87: Begrenzung der Kranbewegung an den Endpunkten,
- Sätze 88, 89, 90, 91, 92, 93: Begrenzung der Greiferbewegung an den Endpunkten unter Berücksichtigung der Position des Krans (Höhe der Förderbänder sowie des Eingangs- bzw. Ausgangspuffers),
- Satz 94: Der Greifer des Krans darf nur an bestimmten Orten deaktiviert werden.

*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 86<<< (einfaches Verbot - Zustand) */*

*"die Bewegung des Krans nach links (zum Zuführband)" darf nicht gleichzeitig "gestartet" sein , wenn "der Kran" "am Zuführband" ist .*

```
AG !( rdy_plc & Crane_at_fbelt & (Crane_to_fbelt) )
```

*/* >>>Satz 87<<< (einfaches Verbot - Zustand) */*

*"die Bewegung des Krans nach rechts (Ablageband)" darf nicht gleichzeitig "gestartet" sein , wenn "der Kran" "am Ablageband" ist .*

```
AG !( rdy_plc & Crane_at_dbelt & (Crane_to_dbelt) )
```

*/* >>>Satz 88<<< (einfaches Verbot - Zustand) */*

*"das Anheben des Krangreifers" darf nicht gleichzeitig "gestartet" sein , wenn "der Greifer des Krans" "oben" ist .*

```
AG !( rdy_plc & Crane_height=0 & (Crane_lift) )
```

*/* >>>Satz 89<<< (einfaches Verbot - Zustand) */*

*"das Absenken des Krangreifers" darf nicht gleichzeitig "gestartet" sein , wenn "der Greifer des Krans" "unten" ist .*

```
AG !( rdy_plc & Crane_height=1.0 & (Crane_lower) )
```

*/* >>>Satz 90<<< (einfaches Verbot - Zustand) */*

*"das Absenken des Krangreifers" darf nicht gleichzeitig "gestartet" sein , wenn "der Kran" "am Ablageband" ist und "der Greifer des Krans" "auf dem Ablageband" ist .*

```
AG !( rdy_plc & Crane_at_dbelt & Crane_height=0.66 &
```

```
(Crane_lower) )
```

*/* >>>Satz 91<<< (einfaches Verbot - Zustand) */*

*"das Absenken des Krangreifers" darf nicht gleichzeitig "gestartet" sein , wenn "der Kran" "am Eingangspuffer" ist und "der Greifer des Krans" "auf dem Eingangspuffer" ist .*

```
AG !( rdy_plc & Crane_at_source & Crane_height=0.85 &
(Crane_lower) )
```

*/* >>>Satz 92<<< (einfaches Verbot - Zustand) */*

*"das Absenken des Krangreifers" darf nicht gleichzeitig "gestartet" sein , wenn "der Kran" "am Ausgangspuffer" ist und "der Greifer des Krans" "auf dem Ausgangspuffer" ist .*

```
AG !( rdy_plc & Crane_at_drain & Crane_height=0.35 &
(Crane_lower) )
```

*/* >>>Satz 93<<< (einfaches Verbot - Zustand) */*

*"das Absenken des Krangreifers" darf nicht gleichzeitig "gestartet" sein , wenn "der Kran" "am Zuführband" ist und "der Greifer des Krans" "auf dem Zuführband" ist .*

```
AG !( rdy_plc & Crane_at_fbelt & Crane_height=0.94 &
(Crane_lower) )
```

*/* >>>Satz 94<<< (einfaches Verbot - selbstbegrenzt) */*

*Solange "der Magnet des Krangreifers" "aktiv" ist und "der Kran" ist "nicht am Ablageband" und "der Kran" ist "nicht am Zuführband" und "der Kran" ist "nicht am Eingangspuffer" und "der Kran" ist "nicht am Ausgangspuffer" , darf "der Magnet des Krangreifers" niemals "deaktiviert" werden .*

```
AG !( rdy_plc & Crane_mag_on & !Crane_at_dbelt &
!Crane_at_fbelt & !Crane_at_source & !Crane_at_drain &
(!Crane_mag_on) )
```

### *A.7.5 Geräteebene 3 - Steuerung des Zuführbands*

Die Steuerung des Zuführbands erhält von der übergeordneten Zellensteuerung Startsignale zur Ausführung bestimmter Aktionen. Durch verschiedene Sensoren und Aktoren ist die Steuerung des Zuführbands mit den Maschinenelementen verbunden.

*Gegebene Daten:*

| **Technische Bezeichnung** | **Variable** | **Datentyp** | **Kommentar** |
|---|---|---|---|
| Meldung „Zuführband für Aufnahme eines Rohteils bereit“ | `Fbelt_rdy_take` | Bool | |
| Bereitschaftsmeldung „Zuführband bereit zum Transport“ | `Fbelt_rdy_transport` | Bool | Zuführband ist zum Transport bereit |
| Startsignal „Rohteil zum Bandende transportieren“ | `Fbelt_run_transport` | Bool | Zuführband soll Teil zum Bandende transportieren |
| Bereitschaftsmeldung „Zuführband bereit zur Übergabe“ | `Fbelt_rdy_give` | Bool | Zuführband ist zur Übergabe bereit |
| Startsignal „Rohteil zum Hubdrehtisch transportieren“ | `Fbelt_run_table` | Bool | Zuführband soll Teil auf Tisch transportie-ren |
| Übergabemeldung des Zuführbands | `Fbelt_done_give` | Bool | Zuführband hat Teil auf Tisch transportiert |

*Definition neuer Daten mit dem SFS-Editor*

Sensor des Zuführbands

| **Technische Bezeichnung** | **SPS-Variable** | **Datentyp** | **Kommentar** |
|---|---|---|---|
| Lichtschranke am Zuführband | `LB_at_fbelt` | Bool | TRUE - Lichtschranke frei |

*Tabelle 28 - Datenebene 4: Sensor des Zuführbands*

```
%%72 -BOOL -VAR
"LB_at_fbelt" : "die Lichtschranke am Ende des Zuführbands"
FALSE_I : "unterbrochen"
TRUE_I : "frei"
```

Aktor des Zuführbands

| Technische Bezeichnung | SPS-Variable | Datentyp | Kommentar |
|---|---|---|---|
| Antriebsmotor Zuführband | FBelt_start | Bool | TRUE - Band läuft |

*Tabelle 29 - Datenebene 4: Aktor des Zuführbands*

```
%%73 -BOOL -VAR
"FBelt_start" : "der Antrieb des Zuführbands"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
TRUE_O : "gestartet"
FALSE_O : "gestoppt"
```

*Informelle Spezifikation:*

Der Steuerung des Zuführbands verbleibt solange in Ruhe, bis durch die Zellensteuerung entweder das Startsignal zum Transport eines Rohteils zum Bandende oder zur Übergabe auf den Hubdrehtisch erfolgt. Danach wird die geforderte Transportroutine bearbeitet. Es sind also zwei separate Transportvorgänge zu unterscheiden, die alternativ voneinander ablaufen können. Die innerhalb dieser Routinen abzuarbeitenden Teilvorgänge sind wiederum unbedingt in der festgelegten Reihenfolge abzuarbeiten, es bietet sich wiederum die Entwicklung einer Ablaufstruktur an.

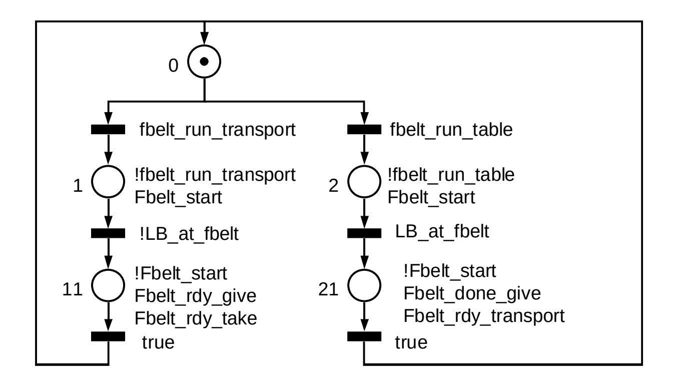
*Defintion neuer Daten mit dem SFS-Editor:*

Interne Variable der Steuerung des Zuführbands

| Technische Bezeichnung | SPS-Variable | Datentyp | Kommentar |
|---|---|---|---|
| Zustandsvariable des Zuführbands | Fbelt_state | Integer | |

*Tabelle 30 - Datenebene 4: Interne Variable der Steuerung des Zuführbands*

```
%%74 -INT -VAR
"Fbelt_state" : "die Zustandsvariable des Zuführbands"
0_I : "0"
1_I : "1"
11_I : "11"
2_I : "2"
21_I : "21"
0_O : "auf 0 gesetzt"
1_O : "auf 1 gesetzt"
11_O : "auf 11 gesetzt"
2_O : "auf 2 gesetzt"
21_O : "auf 21 gesetzt"
```




*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 96<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Zuführbands" "0" ist und "das Startsignal 'Zuführband soll Teil zum Bandende transportieren'" ist "gesetzt" , dann muss "das Startsignal 'Zuführband soll Teil zum Bandende transportieren'" unmittelbar "zurückgesetzt" werden und "die Zustandsvariable des Zuführbands" muss unmittelbar "auf 1 gesetzt" werden und "der Antrieb des Zuführbands" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Fbelt_state=0 & Fbelt_run_transport) ->
A[!rdy_plc U (rdy_plc & !Fbelt_run_transport &
Fbelt_state=1 & FBelt_start) ] )
```

*/* >>>Satz 97<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Zuführbands" "0" ist und "das Startsignal 'Zuführband soll Teil auf Tisch befördern'" ist "gesetzt" , dann muss "das Startsignal 'Zuführband soll Teil auf Tisch befördern'" unmittelbar "zurückgesetzt" werden und "die Zustandsvariable des Zuführbands" muss unmittelbar "auf 2 gesetzt" werden und "der Antrieb des Zuführbands" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Fbelt_state=0 & Fbelt_run_table) ->
A[!rdy_plc U (rdy_plc & !Fbelt_run_table & Fbelt_state=2 &
FBelt_start) ] )
```

*/* >>>Satz 98<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Zuführbands" "1" ist und "die Lichtschranke am Ende des Zuführbands" ist "unterbrochen" , dann muss "der Antrieb des Zuführbands" unmittelbar "gestoppt" werden und "die Zustandsvariable des Zuführbands" muss unmittelbar "auf 11 gesetzt" werden und "die Bereitschaftsmeldung 'Zuführband ist zur Übergabe bereit'" muss unmittelbar "gesetzt" werden und "die Bereitschaftsmeldung 'Zuführband ist zur Übernahme eines Teils bereit'" muss unmittelbar "gesetzt" werden .*

```
AG ( (rdy_in & Fbelt_state=1 & !LB_at_fbelt) -> A[!rdy_plc
U (rdy_plc & !FBelt_start & Fbelt_state=11 &
Fbelt_ready_give & Fbelt_ready_take) ] )
```

*/* >>>Satz 99<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Zuführbands" "2" ist und "die Lichtschranke am Ende des Zuführbands" ist "frei" , dann muss "der Antrieb des Zuführbands" unmittelbar "gestoppt" werden und "die Zustandsvariable des Zuführbands" muss unmittelbar "auf 21 gesetzt" werden und "die Fertigmeldung 'Zuführband hat Teil auf Tisch abgelegt'" muss unmittelbar "gesetzt" werden und "die Bereitschaftsmeldung 'Zuführband ist zum Transport bereit'" muss unmittelbar "gesetzt" werden .*

```
AG ( (rdy_in & Fbelt_state=2 & LB_at_fbelt) -> A[!rdy_plc
U (rdy_plc & !FBelt_start & Fbelt_state=21 &
Fbelt_done_give & Fbelt_ready_transport) ] )
```

*/* >>>Satz 100<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Zuführbands" "11" ist , dann muss "die Zustandsvariable des Zuführbands" unmittelbar "auf 0 gesetzt" werden .*

```
AG ( (rdy_in & Fbelt_state=11) -> A[!rdy_plc U (rdy_plc &
Fbelt_state=0) ] )
```

*/* >>>Satz 101<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Zuführbands" "21" ist , dann muss "die Zustandsvariable des Zuführbands" unmittelbar "auf 0 gesetzt" werden .*

```
AG ( (rdy_in & Fbelt_state=21) -> A[!rdy_plc U (rdy_plc &
Fbelt_state=0) ] )
```

*Informelle Spezifikation weiterer Anforderungen:*

Sicherheitskritische Punkte

- *Satz 102: Übergabe auf Hubdrehtisch*

*Formale Spezifikation mit dem SFS-Editor*

*/* >>>Satz 102<<< (einfaches Verbot - Zustand) */*

*Wenn "die Lichtschranke am Ende des Zuführbands" "unterbrochen" ist und "die Bereitschaftsmeldung 'Hubdrehtisch ist zur Übernahme eines Teils bereit'" ist "nicht gesetzt" , dann darf "der Antrieb des Zuführbands" nicht gleichzeitig "gestartet" sein .*

```
AG !( rdy_plc & !LB_at_fbelt & !Table_ready_take &
(FBelt_start) )
```

### A.7.6 Geräteebene 3 - Steuerung des Hubdrehtischs

Die Steuerung des Hubdrehtischs erhält von der übergeordneten Zellensteuerung Startsignale zur Ausführung bestimmter Aktionen. Durch verschiedene Sensoren und Aktoren ist die Steuerung des Hubdrehtischs mit den Maschinenelementen verbunden.

*Gegebene Daten:*

| Technische Bezeichnung | Variable | Datentyp | Kommentar |
|---|---|---|---|
| Fertigmeldung „Tisch bereit zur Übernahme“ | `Table_rdy_take` | Bool | Tisch ist zu Entgegennahme eines Teils bereit |
| Startsignal „Heben/Drehen“ | `Table_run_lift` | Bool | |
| Startsignal „Senken/Drehen“ | `Table_run_lower` | Bool | |
| Meldung „Hubdrehtisch ist zur Übergabe bereit“ | `Table_rdy_give` | Bool | |

*Definition neuer Daten mit dem SFS-Editor*

Sensoren des Hubdrehtischs

| Technische Bezeichnung | SPS-Variable | Datentyp | Kommentar |
|---|---|---|---|
| Schalter untere Tischposition | `Table_bottom` | Bool | TRUE - Tisch unten |
| Schalter obere Tischposition | `Table_top` | Bool | TRUE - Tisch oben |
| Drehgeber Hubdrehtisch | `Table_angle` | Integer | 0°- Tisch vor Band<br>50°- Tisch vor Roboter<br>85°- Maximum |

*Tabelle 31 - Datenebene 4: Sensoren des Hubdrehtischs*

```
%%75 -BOOL -VAR
"Table_bottom" : "der Hubdrehtisch"
TRUE_I : "unten"
FALSE_I : "nicht unten"

%%76 -BOOL -VAR
"Table_top" : "der Hubdrehtisch"
TRUE_I : "oben"
FALSE_I : "nicht oben"

%%77 -INT -VAR
"Table_angle" : "der Hubdrehtisch"
50_I : "vor dem Roboter"
```

```
0_I : "vor dem Zuführband"
```

Aktoren des Hubdrehtischs

| Technische Bezeichnung | SPS-Variable | Datentyp | Kommentar |
|---|---|---|---|
| Motor Tischdrehung (L/R) | `Table_left` | Bool | TRUE - Tisch dreht nach links |
| | `Table_right` | Bool | TRUE - Tisch dreht nach rechts |
| Motor Tischhöhe (L/R) | `Table_upward` | Bool | TRUE - Tisch hebt sich |
| | `Table_downward` | Bool | TRUE - Tisch senkt sich |

*Tabelle 32 - Datenebene 4: Aktoren des Hubdrehtischs*

```
%%78 -BOOL -VAR
"Table_left" : "die Linksdrehung des Hubdrehtischs"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
TRUE_O : "gestartet"
FALSE_O : "gestoppt"

%%79-BOOL -VAR
"Table_right" : "die Rechtsdrehung des Hubdrehtischs"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
TRUE_O : "gestartet"
FALSE_O : "gestoppt"

%%80 -BOOL -VAR
"Table_upward" : "das Anheben des Hubdrehtischs"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
TRUE_O : "gestartet"
FALSE_O : "gestoppt"

%%81 -BOOL -VAR
"Table_downward" : "das Absenken des Hubdrehtischs"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
TRUE_O : "gestartet"
FALSE_O : "gestoppt"
```

*Informelle Spezifikation:*

Die Steuerung des Hubdrehtischs verbleibt solange in Ruhe, bis durch die Zellensteuerung entweder das Startsignal zum Heben und Linksdrehen oder zum Absenken und Rechtsdrehen erfolgt. Danach wird die geforderte Bewegungsroutine bearbeitet. Es sind zwei separate Routinen zu unterscheiden, die alternativ voneinander ablaufen können. Die innerhalb dieser Routinen abzuarbeitenden Teilvorgänge sind wiederum unbedingt in der festgelegten Reihenfolge abzuarbeiten, es bietet sich wiederum die Entwicklung einer Ablaufstruktur an.

*Definition neuer Daten mit dem SFS-Editor:*

Interne Variablen der Steuerung des Hubdrehtischs

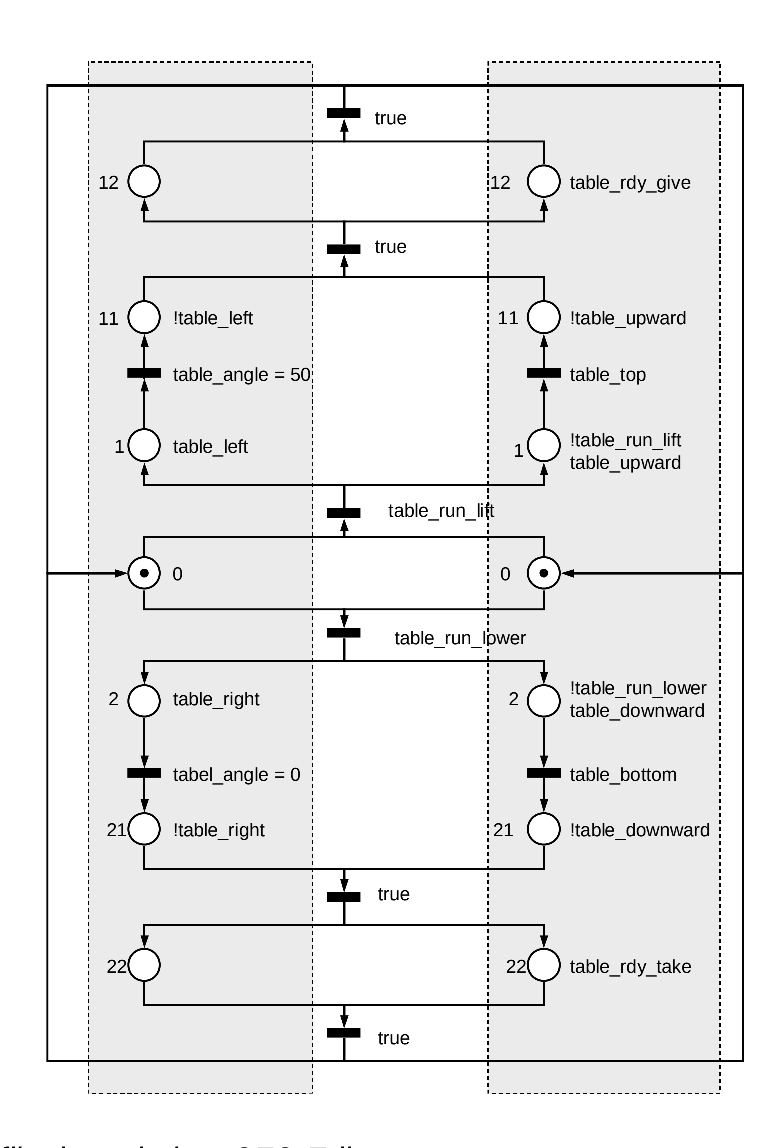
| **Technische Bezeichnung** | **SPS-Variable** | **Datentyp** | **Kommentar** |
|---|---|---|---|
| Zustandsvariable des Hubdrehtischs (Drehung) | `Table_state1` | Integer | |
| Zustandsvariable des Hubdrehtischs (Höhe) | `Table_state2` | Integer | |

*Tabelle 33 - Datenebene 4: Interne Variablen der Steuerung des Hubdrehtischs*

```
%%82 -INT -VAR
"Table_state1" : "die Zustandsvariable der Hubtischdrehung"
0_I : "0"
1_I : "1"
2_I : "2"
11_I : "11"
12_I : "12"
21_I : "21"
22_I : "22"
0_O : "auf 0 gesetzt"
1_O : "auf 1 gesetzt"
2_O : "auf 2 gesetzt"
11_O : "auf 11 gesetzt"
12_O : "auf 12 gesetzt"
21_O : "auf 21 gesetzt"
22_O : "auf 22 gesetzt"

%%83 -INT -VAR
"Table_state2" : "die Zustandsvariable der Hubtischhöhe"
0_I : "0"
1_I : "1"
2_I : "2"
11_I : "11"
12_I : "12"
21_I : "21"
22_I : "22"
0_O : "auf 0 gesetzt"
1_O : "auf 1 gesetzt"
2_O : "auf 2 gesetzt"
```

```
11_O : "auf 11 gesetzt"
12_O : "auf 12 gesetzt"
21_O : "auf 21 gesetzt"
22_O : "auf 22 gesetzt"
```




*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 105<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable der Hubtischhöhe" "0" ist und "die Zustandsvariable der Hubtischdrehung" ist "0" und "das Startsignal 'Hubdrehtisch heben und links drehen'" ist "gesetzt" , dann muss "das Startsignal 'Hubdrehtisch heben und links drehen'" unmittelbar "zurückgesetzt" werden und "die Zustandsvariable der Hubtischhöhe" muss unmittelbar "auf 1 gesetzt" werden und "die Zustandsvariable der Hubtischdrehung" muss unmittelbar "auf 1 gesetzt" werden und "das Anheben des Hubdrehtischs" muss unmittelbar "gestartet" werden und "die Linksdrehung des Hubdrehtischs" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Table_state2=0 & Table_state1=0 &
Table_run_lift) -> A[!rdy_plc U (rdy_plc & !Table_run_lift
```

```
& Table_state2=1 & Table_state1=1 & Table_upward &
Table_left) ] )
```

*/* >>>Satz 106<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable der Hubtischhöhe" "0" ist und "die Zustandsvariable der Hubtischdrehung" ist "0" und "das Startsignal 'Hubdrehtisch senken und rechts drehen'" ist "gesetzt" , dann muss "das Startsignal 'Hubdrehtisch senken und rechts drehen'" unmittelbar "zurückgesetzt" werden und "die Zustandsvariable der Hubtischhöhe" muss unmittelbar "auf 2 gesetzt" werden und "die Zustandsvariable der Hubtischdrehung" muss unmittelbar "auf 2 gesetzt" werden und "das Absenken des Hubdrehtischs" muss unmittelbar "gestartet" werden und "die Rechtsdrehung des Hubdrehtischs" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Table_state2=0 & Table_state1=0 &
Table_run_lower) -> A[!rdy_plc U (rdy_plc &
!Table_run_lower & Table_state2=2 & Table_state1=2 &
Table_downward & Table_right) ] )
```

*/* >>>Satz 107<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable der Hubtischhöhe" "1" ist und "der Hubdrehtisch" ist "oben" , dann muss "das Anheben des Hubdrehtischs" unmittelbar "gestoppt" werden und "die Zustandsvariable der Hubtischhöhe" muss unmittelbar "auf 11 gesetzt" werden .*

```
AG ( (rdy_in & Table_state2=1 & Table_top) -> A[!rdy_plc U
(rdy_plc & !Table_upward & Table_state2=11) ] )
```

*/* >>>Satz 108<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable der Hubtischdrehung" "1" ist und "der Hubdrehtisch" ist "vor dem Roboter" , dann muss "die Linksdrehung des Hubdrehtischs" unmittelbar "gestoppt" werden und "die Zustandsvariable der Hubtischdrehung" muss unmittelbar "auf 11 gesetzt" werden .*

```
AG ( (rdy_in & Table_state1=1 & Table_angle=50) ->
A[!rdy_plc U (rdy_plc & !Table_left & Table_state1=11) ] )
```

*/* >>>Satz 109<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable der Hubtischhöhe" "11" ist und "die Zustandsvariable der Hubtischdrehung" ist "11" , dann muss "die Bereitschaftsmeldung 'Hubdrehtisch ist zur Übergabe eines Teils bereit" unmittelbar "gesetzt" werden und "die Zustandsvariable der Hubtischhöhe" muss unmittelbar "auf 12 gesetzt" werden und "die Zustandsvariable der Hubtischdrehung" muss unmittelbar "auf 12 gesetzt" werden .*

```
AG ( (rdy_in & Table_state2=11 & Table_state1=11) ->
```

```
A[!rdy_plc U (rdy_plc & Table_ready_give & Table_state2=12
& Table_state1=12) ] )
```

*/* >>>Satz 110<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable der Hubtischhöhe" "12" ist und "die Zustandsvariable der Hubtischdrehung" ist "12" , dann "die Zustandsvariable der Hubtischhöhe" muss unmittelbar "auf 0 gesetzt" werden und "die Zustandsvariable der Hubtischdrehung" muss unmittelbar "auf 0 gesetzt" werden .*

```
AG ( (rdy_in & Table_state2=12 & Table_state1=12) ->
A[!rdy_plc U (rdy_plc & Table_state2=0 & Table_state1=0) ]
)
```

*/* >>>Satz 111<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable der Hubtischhöhe" "2" ist und "der Hubdrehtisch" ist "unten" , dann muss "das Absenken des Hubdrehtischs" unmittelbar "gestoppt" werden und "die Zustandsvariable der Hubtischhöhe" muss unmittelbar "auf 21 gesetzt" werden .*

```
AG ( (rdy_in & Table_state2=2 & Table_bottom) ->
A[!rdy_plc U (rdy_plc & !Table_downward & Table_state2=21)
] )
```

*/* >>>Satz 112<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable der Hubtischdrehung" "2" ist und "der Hubdrehtisch" ist "vor dem Zuführband" , dann muss "die Rechtsdrehung des Hubdrehtischs" unmittelbar "gestoppt" werden und "die Zustandsvariable der Hubtischdrehung" muss unmittelbar "auf 21 gesetzt" werden .*

```
AG ( (rdy_in & Table_state1=2 & Table_angle=0) ->
A[!rdy_plc U (rdy_plc & !Table_right & Table_state1=21) ]
)
```

*/* >>>Satz 113<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable der Hubtischhöhe" "21" ist und "die Zustandsvariable der Hubtischdrehung" ist "21" , dann muss "die Bereitschaftsmeldung 'Hubdrehtisch ist zur Übernahme eines Teils bereit'" unmittelbar "gesetzt" werden und "die Zustandsvariable der Hubtischhöhe" muss unmittelbar "auf 22 gesetzt" werden und "die Zustandsvariable der Hubtischdrehung" muss unmittelbar "auf 22 gesetzt" werden .*

```
AG ( (rdy_in & Table_state2=21 & Table_state1=21) ->
A[!rdy_plc U (rdy_plc & Table_ready_take & Table_state2=22
& Table_state1=22) ] )
```

*/* >>>Satz 114<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable der Hubtischhöhe" "22" ist und "die Zustandsvariable der Hubtischdrehung" ist "22" , dann "die Zustandsvariable der Hubtischhöhe" muss unmittelbar "auf 0 gesetzt" werden und "die Zustandsvariable der Hubtischdrehung" muss unmittelbar "auf 0 gesetzt" werden .*

```
AG ( (rdy_in & Table_state2=22 & Table_state1=22) ->
A[!rdy_plc U (rdy_plc & Table_state2=0 & Table_state1=0) ]
)
```

*Informelle Spezifikation weiterer Eigenschaften:*

Es muss eine Bewegungsbegrenzung bezüglich der Tischhöhe erfolgen.

*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 115<<< (einfaches Verbot - Zustand) */*

*Wenn "der Hubdrehtisch" "oben" ist , dann darf "das Anheben des Hubdrehtischs" nicht gleichzeitig "gestartet" sein .*

```
AG !( rdy_plc & Table_top & (Table_upward) )
```

*/* >>>Satz 116<<< (einfaches Verbot - Zustand) */*

*Wenn "der Hubdrehtisch" "unten" ist , dann darf "das Absenken des Hubdrehtischs" nicht gleichzeitig "gestartet" sein .*

```
AG !( rdy_plc & Table_bottom & (Table_downward) )
```

### *A.7.7 Geräteebene 3 - Steuerung des Roboters*

Die Steuerung des Roboters erhält von der übergeordneten Zellensteuerung Startsignale zur Ausführung bestimmter Aktionen. Durch verschiedene Sensoren und Aktoren ist die Steuerung des Zuführbands mit den Maschinenelementen verbunden.

*Gegebene Daten:*

| Technische Bezeichnung | Variable | Datentyp | Kommentar |
|---|---|---|---|
| Fertigmeldung „Roboter hat Teil vom Tisch genommen“ | `Robot_done_take_table` | Bool | |
| Startsignal „Presse beladen“ | `Robot_run_load` | Bool | |
| Startsignal „Presse entladen“ | `Robot_run_unload` | Bool | |
| Startsignal „Mixbetrieb“ | `Robot_run_mix` | Bool | |
| Meldung „Roboter wartet vor Presse“ | `Robot_wait_mix` | Bool | |
| Startsignal „Weiter im Mix“ | `Robot_run_cont` | Bool | |
| Fertigmeldung „Presse beladen“ | `Robot_done_load` | Bool | |
| Fertigmeldung „Presse entladen“ | `Robot_done_unload` | Bool | |
| Fertigmeldung „Roboter hat Band beladen“ | `Robot_done_dbelt` | Bool | |

*Definition neuer Daten mit dem SFS-Editor:*

Sensoren des Roboters

| Technische Bezeichnung | SPS-Variable | Datentyp | Kommentar |
|---|---|---|---|
| Wegmesssystem Roboterarm 1 | `Arm1_ext` | Integer | 0,00 - Arm hinten<br>0,37 - Arm an Presse<br>0,52 - Arm am Hubdrehtisch<br>0,65 - Arm in Presse<br>1,00 Arm vorn |
| Wegmesssystem Roboterarm 2 | `Arm2_ext` | Integer | 0,00 - Arm hinten<br>0,57 - Arm am Ablageband<br>0,79 - Arm in Presse<br>1,00 Arm vorn |
| Drehgeber Roboter | `Robot_angle` | Integer | -95°- Minimum<br>-90°- Arm 1 vor Presse<br>-70°- Arm 2 über Abl.-band |

| | | | |
|---|---|---|---|
| | | | 0°- Arm 2 an Presse<br>50°- Arm 1 an Hubdrehtisch<br>65°- Maximum |

*Tabelle 34 - Datenebene 4: Sensoren des Roboters*

```
%%84 -INT -VAR
"Arm1_ext" : "der Roboterarm 1"
0.0_I : "hinten"
0.37_I : "an der Presse"
0.52_I : "am Hubdrehtisch"
0.65_I : "in der Presse"
1.0_I : "vorn"

%%85 -INT -VAR
"Arm2_ext" : "der Roboterarm 2"
0.0_I : "hinten"
0.57_I : "am Ablageband"
0.79_I : "in der Presse"
1.0_I : "vorn"

%%86 -INT -VAR
"Robot_angle" : "der Roboter"
-90_I : "mit dem Arm 1 vor der Presse"
50_I : "mit dem Arm 1 vor dem Hubdrehtisch"
0_I : "mit dem Arm 2 vor der Presse"
-70_I : "mit dem Arm 2 vor dem Ablageband"
```

Aktoren des Roboters

| **Technische Bezeichnung** | **SPS-Variable** | **Datentyp** | **Kommentar** |
|---|---|---|---|
| Motor Roboterarm 1 (L/R) | `Arm1_forward` | Bool | TRUE - Arm 1 fährt aus |
| | `Arm1_backward` | Bool | TRUE - Arm 1 fährt ein |
| Motor Roboterarm 2 (L/R) | `Arm2_forward` | Bool | TRUE - Arm 2 fährt aus |
| | `Arm2_backward` | Bool | TRUE - Arm 2 fährt ein |
| Motor Roboterdre-hung (L/R) | `Robot_left` | Bool | TRUE - Roboter dreht nach links |
| | `Robot_right` | Bool | TRUE - Roboter dreht nach rechts |
| Greifer Roboterarm 1 | `Arm1_mag_on` | Bool | TRUE - Magnet eingeschaltet |
| Greifer Roboterarm 2 | `Arm2_mag_on` | Bool | TRUE - Magnet eingeschaltet |

*Tabelle 35 - Datenebene 4: Aktoren des Roboters*

```
%%87 -BOOL -VAR
"Arm1_forward" : "das Ausfahren des Roboterarms 1"
TRUE_O : "gestartet"
FALSE_O : "gestoppt"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"

%%88 -BOOL -VAR
"Arm1_backward" : "das Einziehen des Roboterarms 1"
TRUE_O : "gestartet"
FALSE_O : "gestoppt"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"

%%89 -BOOL -VAR
"Arm2_forward" : "das Ausfahren des Roboterarms 2"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
TRUE_O : "gestartet"
FALSE_O : "gestoppt"

%%90 -BOOL -VAR
"Arm2_backward" : "das Einziehen des Roboterarms 2"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
TRUE_O : "gestartet"
FALSE_O : "gestoppt"

%%91 -BOOL -VAR
"Arm1_mag_on" : "der Magnetgreifer des Roboterarms 1"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
TRUE_O : "aktiviert"
FALSE_O : "deaktiviert"

%%92 -BOOL -VAR
"Arm2_mag_on" : "der Magnetgreifer des Roboterarms 2"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
TRUE_O : "aktiviert"
FALSE_O : "deaktiviert"

%%93 -BOOL -VAR
"Robot_left" : "die Drehung des Roboters nach links"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
TRUE_O : "gestartet"
FALSE_O : "gestoppt"

%%94 -BOOL -VAR
"Robot_right" : "die Drehung des Roboters nach rechts"
```

```
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
TRUE_O : "gestartet"
FALSE_O : "gestoppt"
```

*Informelle Spezifikation:*

Die Steuerung des Roboters verbleibt solange in Ruhe, bis durch die Zellensteuerung entweder das Startsignal zur ausschließlichen Beladung der Presse, zur ausschließlichen Entladung der Presse oder zur koordinierten Be- und Entladung der Presse erfolgt. Danach wird die geforderte Transportroutine bearbeitet. Es sind drei separate Transportvorgänge zu unterscheiden, die alternativ voneinander ablaufen können. Die innerhalb dieser Routinen abzuarbeitenden Teilvorgänge sind unbedingt in der festgelegten Reihenfolge abzuarbeiten, es bietet sich die Entwicklung einer Ablaufstruktur an.

*Definition neuer Daten mit dem SFS-Editor:*

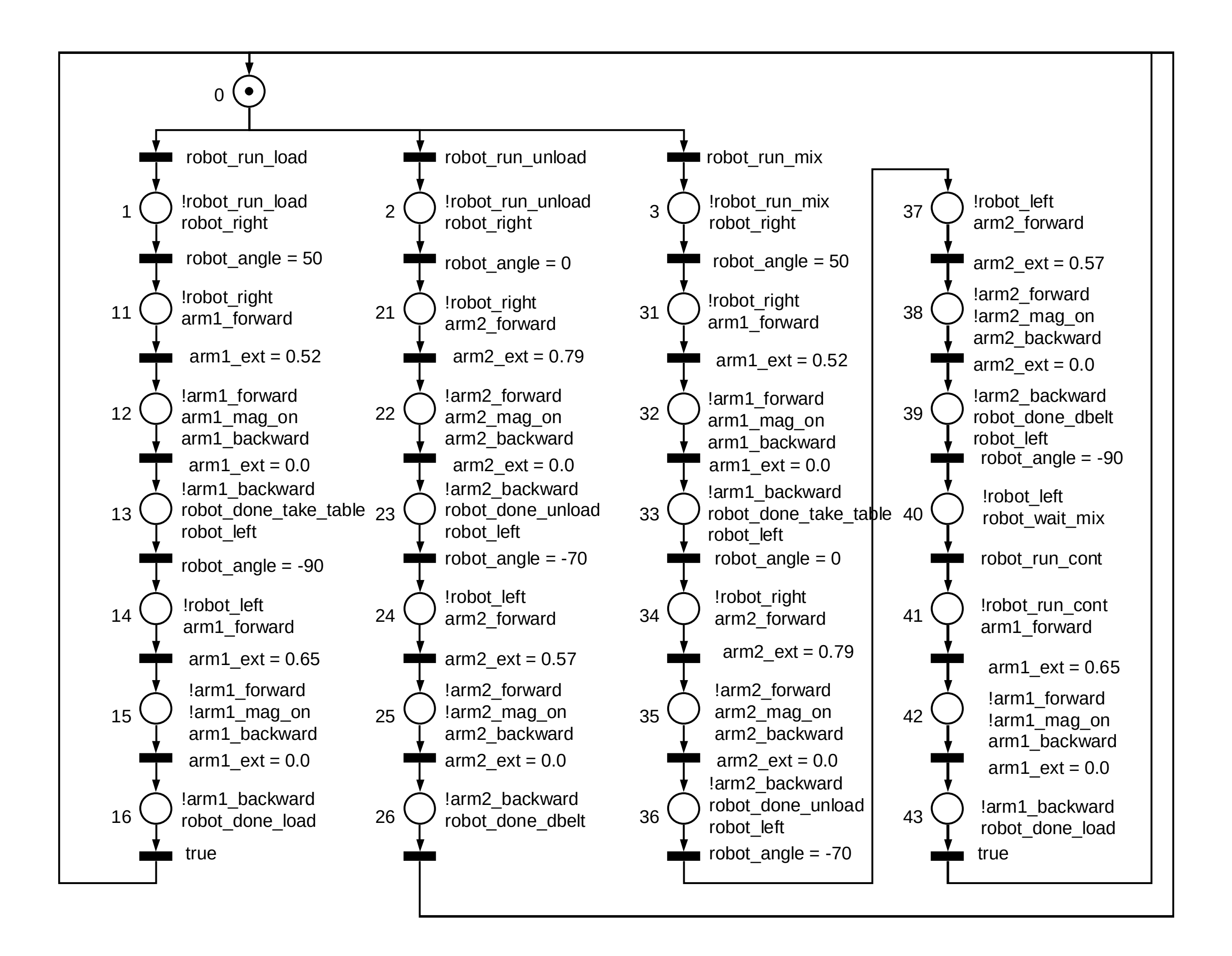
Interne Variable der Steuerung des Roboters

| **Technische Bezeichnung** | **SPS-Variable** | **Datentyp** | **Kommentar** |
|---|---|---|---|
| Zustandsvariable des Roboters | Robot_state | Integer | |

*Tabelle 36 - Datenebene 4: Interne Variable der Steuerung des Roboters*

```
%%95 -INT -VAR
"Robot_state" : "die Zustandsvariable des Roboters"
0_I : "0"
1_I : "1"
2_I : "2"
3_I : "3"
11_I : "11"
12_I : "12"
13_I : "13"
14_I : "14"
15_I : "15"
16_I : "16"
21_I : "21"
22_I : "22"
23_I : "23"
24_I : "24"
25_I : "25"
26_I : "26"
31_I : "31"
32_I : "32"
33_I : "33"
34_I : "34"
35_I : "35"
36_I : "36"
37_I : "37"
```

```
38_I : "38"
39_I : "39"
40_I : "40"
41_I : "41"
42_I : "42"
43_I : "43"
0_O : "auf 0 gesetzt"
1_O : "auf 1 gesetzt"
2_O : "auf 2 gesetzt"
3_O : "auf 3 gesetzt"
11_O : "auf 11 gesetzt"
12_O : "auf 12 gesetzt"
13_O : "auf 13 gesetzt"
14_O : "auf 14 gesetzt"
15_O : "auf 15 gesetzt"
16_O : "auf 16 gesetzt"
21_O : "auf 21 gesetzt"
22_O : "auf 22 gesetzt"
23_O : "auf 23 gesetzt"
24_O : "auf 24 gesetzt"
25_O : "auf 25 gesetzt"
26_O : "auf 26 gesetzt"
31_O : "auf 31 gesetzt"
32_O : "auf 32 gesetzt"
33_O : "auf 33 gesetzt"
34_O : "auf 34 gesetzt"
35_O : "auf 35 gesetzt"
36_O : "auf 36 gesetzt"
37_O : "auf 37 gesetzt"
38_O : "auf 38 gesetzt"
39_O : "auf 39 gesetzt"
40_O : "auf 40 gesetzt"
41_O : "auf 41 gesetzt"
42_O : "auf 42 gesetzt"
43_O : "auf 43 gesetzt"
```




*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 118<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "0" ist und "das Startsignal 'Roboter belädt Presse'" ist "gesetzt" , dann muss "das Startsignal 'Roboter belädt Presse'" unmittelbar "zurückgesetzt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 1 gesetzt" werden und "die Drehung des Roboters nach rechts" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Robot_state=0 & Robot_run_load) ->
A[!rdy_plc U (rdy_plc & !Robot_run_load & Robot_state=1 &
Robot_right) ] )
```

*/* >>>Satz 119<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "0" ist und "das Startsignal 'Roboter entlädt Presse'" ist "gesetzt" , dann muss "das Startsignal 'Roboter entlädt Presse'" unmittelbar "zurückgesetzt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 2 gesetzt" werden und "die Drehung des Roboters nach rechts" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Robot_state=0 & Robot_run_unload) ->
A[!rdy_plc U (rdy_plc & !Robot_run_unload & Robot_state=2
& Robot_right) ] )
```

*/* >>>Satz 120<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "0" ist und "das Startsignal 'Roboter be- und entlädt Presse'" ist "gesetzt" , dann muss "das Startsignal 'Roboter be- und entlädt Presse'" unmittelbar "zurückgesetzt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 3 gesetzt" werden und "die Drehung des Roboters nach rechts" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Robot_state=0 & Robot_run_mix) ->
A[!rdy_plc U (rdy_plc & !Robot_run_mix & Robot_state=3 &
Robot_right) ] )
```

*/* >>>Satz 121<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "1" ist und "der Roboter" ist "mit dem Arm 1 vor dem Hubdrehtisch" , dann muss "die Drehung des Roboters nach rechts" unmittelbar "gestoppt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 11 gesetzt" werden und "das Ausfahren des Roboterarms 1" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Robot_state=1 & Robot_angle=50) ->
A[!rdy_plc U (rdy_plc & !Robot_right & Robot_state=11 &
Arm1_forward) ] )
```

*/* >>>Satz 122<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "11" ist und "der Roboterarm 1" ist "am Hubdrehtisch" , dann muss "das Ausfahren des Roboterarms 1" unmittelbar "gestoppt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 12 gesetzt" werden und "der Magnetgreifer des Roboterarms 1" muss unmittelbar "aktiviert" werden und "das Einziehen des Roboterarms 1" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Robot_state=11 & Arm1_ext=0.52) ->
A[!rdy_plc U (rdy_plc & !Arm1_forward & Robot_state=12 &
Arm1_mag_on & Arm1_backward) ] )
```

*/* >>>Satz 123<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "12" ist und "der Roboterarm 1" ist "hinten" , dann muss "das Einziehen des Roboterarms 1" unmittelbar "gestoppt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 13 gesetzt" werden und "die Fertigmeldung 'Roboter hat Teil vom Tisch genommen'" muss unmittelbar "gesetzt" werden und "die Drehung des Roboters nach links" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Robot_state=12 & Arm1_ext=0.0) ->
A[!rdy_plc U (rdy_plc & !Arm1_backward & Robot_state=13 &
Robot_done_take_table & Robot_left) ] )
```

*/* >>>Satz 124<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "13" ist und "der Roboter" ist "mit dem Arm 1 vor der Presse" , dann muss "die Drehung des Roboters nach links" unmittelbar "gestoppt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 14 gesetzt" werden und "das Ausfahren des Roboterarms 1" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Robot_state=13 & Robot_angle=-90) ->
A[!rdy_plc U (rdy_plc & !Robot_left & Robot_state=14 &
Arm1_forward) ] )
```

*/* >>>Satz 125<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "14" ist und "der Roboterarm 1" ist "in der Presse" , dann muss "das Ausfahren des Roboterarms 1" unmittelbar "gestoppt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 15 gesetzt" werden und "der Magnetgreifer des Roboterarms 1" muss unmittelbar "deaktiviert" werden und "das Einziehen des Roboterarms 1" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Robot_state=14 & Arm1_ext=0.65) ->
A[!rdy_plc U (rdy_plc & !Arm1_forward & Robot_state=15 &
!Arm1_mag_on & Arm1_backward) ] )
```

*/* >>>Satz 126<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "15" ist und "der Roboterarm 1" ist "hinten" , dann muss "das Einziehen des Roboterarms 1" unmittelbar "gestoppt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 16 gesetzt" werden und "die Fertigmeldung 'Roboter hat Teil in Presse gelegt'" muss unmittelbar "gesetzt" werden .*

```
AG ( (rdy_in & Robot_state=15 & Arm1_ext=0.0) ->
A[!rdy_plc U (rdy_plc & !Arm1_backward & Robot_state=16 &
Robot_done_load) ] )
```

*/* >>>Satz 127<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "16" ist , dann "die Zustandsvariable des Roboters" muss unmittelbar "auf 0 gesetzt" werden .*

```
AG ( (rdy_in & Robot_state=16) -> A[!rdy_plc U (rdy_plc &
Robot_state=0) ] )
```

*/* >>>Satz 128<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "2" ist und "der Roboter" ist "mit dem Arm 2 vor der Presse" , dann muss "die Drehung des Roboters nach rechts" unmittelbar "gestoppt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 21 gesetzt" werden und "das Ausfahren des Roboterarms 2" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Robot_state=2 & Robot_angle=0) ->
A[!rdy_plc U (rdy_plc & !Robot_right & Robot_state=21 &
Arm2_forward) ] )
```

*/* >>>Satz 129<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "21" ist und "der Roboterarm 2" ist "in der Presse" , dann muss "das Ausfahren des Roboterarms 2" unmittelbar "gestoppt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 22 gesetzt" werden und "der Magnetgreifer des Roboterarms 2" muss unmittelbar "aktiviert" werden und "das Einziehen des Roboterarms 2" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Robot_state=21 & Arm2_ext=0.79) ->
A[!rdy_plc U (rdy_plc & !Arm2_forward & Robot_state=22 &
Arm2_mag_on & Arm2_backward) ] )
```

*/* >>>Satz 130<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "22" ist und "der Roboterarm 2" ist "hinten" , dann muss "das Einziehen des Roboterarms 2" unmittelbar "gestoppt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 23 gesetzt" werden und "die Fertigmeldung 'Roboter hat Teil aus Presse genommen'" muss unmittelbar "gesetzt" werden und "die Drehung des Roboters nach links" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Robot_state=22 & Arm2_ext=0.0) ->
A[!rdy_plc U (rdy_plc & !Arm2_backward & Robot_state=23 &
Robot_done_unload & Robot_left) ] )
```

*/* >>>Satz 131<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "23" ist und "der Roboter" ist "mit dem Arm 2 vor dem Ablageband" , dann muss "die Drehung des Roboters nach links" unmittelbar "gestoppt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 24 gesetzt" werden und "das Ausfahren des Roboterarms 2" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Robot_state=23 & Robot_angle=-70) ->
A[!rdy_plc U (rdy_plc & !Robot_left & Robot_state=24 &
Arm2_forward) ] )
```

*/* >>>Satz 132<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "24" ist und "der Roboterarm 2" ist "am Ablageband" , dann muss "das Ausfahren des Roboterarms 2" unmittelbar "gestoppt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 25 gesetzt" werden und "der Magnetgreifer des Roboterarms 2" muss unmittelbar "deaktiviert" werden und "das Einziehen des Roboterarms 2" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Robot_state=24 & Arm2_ext=0.57) ->
A[!rdy_plc U (rdy_plc & !Arm2_forward & Robot_state=25 &
!Arm2_mag_on & Arm2_backward) ] )
```

*/* >>>Satz 133<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "25" ist und "der Roboterarm 2" ist "hinten" , dann muss "das Einziehen des Roboterarms 2" unmittelbar "gestoppt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 26 gesetzt" werden und "die Fertigmeldung 'Roboter hat Teil auf Ablageband gelegt'" muss unmittelbar "gesetzt" werden .*

```
AG ( (rdy_in & Robot_state=25 & Arm2_ext=0.0) ->
A[!rdy_plc U (rdy_plc & !Arm2_backward & Robot_state=26 &
Robot_done_dbelt) ] )
```

*/* >>>Satz 134<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "26" ist , dann "die Zustandsvariable des Roboters" muss unmittelbar "auf 0 gesetzt" werden .*

```
AG ( (rdy_in & Robot_state=26) -> A[!rdy_plc U (rdy_plc &
Robot_state=0) ] )
```

*/* >>>Satz 135<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "3" ist und "der Roboter" ist "mit dem Arm 1 vor dem Hubdrehtisch" , dann muss "die Drehung des Roboters nach rechts" unmittelbar "gestoppt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 31 gesetzt" werden und "das Ausfahren des Roboterarms 1" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Robot_state=3 & Robot_angle=50) ->
A[!rdy_plc U (rdy_plc & !Robot_right & Robot_state=31 &
Arm1_forward) ] )
```

*/* >>>Satz 136<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "31" ist und "der Roboterarm 1" ist "am Hubdrehtisch" , dann muss "das Ausfahren des Roboterarms 1" unmittelbar "gestoppt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 32 gesetzt" werden und "der Magnetgreifer des Roboterarms 1" muss unmittelbar*

*"aktiviert" werden und "das Einziehen des Roboterarms 1" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Robot_state=31 & Arm1_ext=0.52) ->
A[!rdy_plc U (rdy_plc & !Arm1_forward & Robot_state=32 &
Arm1_mag_on & Arm1_backward) ] )
```

*/* >>>Satz 137<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "32" ist und "der Roboterarm 1" ist "hinten" , dann muss "das Einziehen des Roboterarms 1" unmittelbar "gestoppt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 33 gesetzt" werden und "die Fertigmeldung 'Roboter hat Teil vom Tisch genommen'" muss unmittelbar "gesetzt" werden und "die Drehung des Roboters nach links" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Robot_state=32 & Arm1_ext=0.0) ->
A[!rdy_plc U (rdy_plc & !Arm1_backward & Robot_state=33 &
Robot_done_take_table & Robot_left) ] )
```

*/* >>>Satz 138<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "33" ist und "der Roboter" ist "mit dem Arm 2 vor der Presse" , dann muss "die Drehung des Roboters nach links" unmittelbar "gestoppt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 34 gesetzt" werden und "das Ausfahren des Roboterarms 2" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Robot_state=33 & Robot_angle=0) ->
A[!rdy_plc U (rdy_plc & !Robot_left & Robot_state=34 &
Arm2_forward) ] )
```

*/* >>>Satz 139<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "34" ist und "der Roboterarm 2" ist "in der Presse" , dann muss "das Ausfahren des Roboterarms 2" unmittelbar "gestoppt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 35 gesetzt" werden und "der Magnetgreifer des Roboterarms 2" muss unmittelbar "aktiviert" werden und "das Einziehen des Roboterarms 2" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Robot_state=34 & Arm2_ext=0.79) ->
A[!rdy_plc U (rdy_plc & !Arm2_forward & Robot_state=35 &
Arm2_mag_on & Arm2_backward) ] )
```

*/* >>>Satz 140<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "35" ist und "der Roboterarm 2" ist "hinten" , dann muss "das Einziehen des Roboterarms 2" unmittelbar "gestoppt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 36 gesetzt"*

*werden und "die Fertigmeldung 'Roboter hat Teil aus Presse genommen'" muss unmittelbar "gesetzt" werden und "die Drehung des Roboters nach links" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Robot_state=35 & Arm2_ext=0.0) ->
A[!rdy_plc U (rdy_plc & !Arm2_backward & Robot_state=36 &
Robot_done_unload & Robot_left) ] )
```

*/* >>>Satz 141<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "36" ist und "der Roboter" ist "mit dem Arm 2 vor dem Ablageband" , dann muss "die Drehung des Roboters nach links" unmittelbar "gestoppt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 37 gesetzt" werden und "das Ausfahren des Roboterarms 2" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Robot_state=36 & Robot_angle=-70) ->
A[!rdy_plc U (rdy_plc & !Robot_left & Robot_state=37 &
Arm2_forward) ] )
```

*/* >>>Satz 142<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "37" ist und "der Roboterarm 2" ist "am Ablageband" , dann muss "das Ausfahren des Roboterarms 2" unmittelbar "gestoppt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 38 gesetzt" werden und "der Magnetgreifer des Roboterarms 2" muss unmittelbar "deaktiviert" werden und "das Einziehen des Roboterarms 2" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Robot_state=37 & Arm2_ext=0.57) ->
A[!rdy_plc U (rdy_plc & !Arm2_forward & Robot_state=38 &
!Arm2_mag_on & Arm2_backward) ] )
```

*/* >>>Satz 143<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "38" ist und "der Roboterarm 2" ist "hinten" , dann muss "das Einziehen des Roboterarms 2" unmittelbar "gestoppt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 39 gesetzt" werden und "die Fertigmeldung 'Roboter hat Teil auf Ablageband gelegt'" muss unmittelbar "gesetzt" werden und "die Drehung des Roboters nach links" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Robot_state=38 & Arm2_ext=0.0) ->
A[!rdy_plc U (rdy_plc & !Arm2_backward & Robot_state=39 &
Robot_done_dbelt & Robot_left) ] )
```

*/* >>>Satz 144<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "39" ist und "der Roboter" ist "mit dem Arm 1 vor der Presse" , dann muss "die Drehung des Roboters nach links" unmittelbar*

*"gestoppt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 40 gesetzt" werden und "die Bereitschaftsmeldung 'Roboter wartet vor Presse'" muss unmittelbar "gesetzt" werden .*

```
AG ( (rdy_in & Robot_state=39 & Robot_angle=-90) ->
A[!rdy_plc U (rdy_plc & !Robot_left & Robot_state=40 &
Robot_wait_mix) ] )
```

*/* >>>Satz 145<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "40" ist und "das Startsignal 'Roboter weiter im Mix"" ist "gesetzt" , dann muss "das Startsignal 'Roboter weiter im Mix"" unmittelbar "zurückgesetzt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 41 gesetzt" werden und "das Ausfahren des Roboterarms 1" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Robot_state=40 & Robot_run_cont) ->
A[!rdy_plc U (rdy_plc & !Robot_run_cont & Robot_state=41 &
Arm1_forward) ] )
```

*/* >>>Satz 146<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "41" ist und "der Roboterarm 1" ist "in der Presse" , dann muss "das Ausfahren des Roboterarms 1" unmittelbar "gestoppt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 42 gesetzt" werden und "der Magnetgreifer des Roboterarms 1" muss unmittelbar "deaktiviert" werden und "das Einziehen des Roboterarms 1" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Robot_state=41 & Arm1_ext=0.65) ->
A[!rdy_plc U (rdy_plc & !Arm1_forward & Robot_state=42 &
!Arm1_mag_on & Arm1_backward) ] )
```

*/* >>>Satz 147<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "42" ist und "der Roboterarm 1" ist "hinten" , dann muss "das Einziehen des Roboterarms 1" unmittelbar "gestoppt" werden und "die Zustandsvariable des Roboters" muss unmittelbar "auf 43 gesetzt" werden und "die Fertigmeldung 'Roboter hat Teil in Presse gelegt'" muss unmittelbar "gesetzt" werden .*

```
AG ( (rdy_in & Robot_state=42 & Arm1_ext=0.0) ->
A[!rdy_plc U (rdy_plc & !Arm1_backward & Robot_state=43 &
Robot_done_load) ] )
```

*/* >>>Satz 148<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Roboters" "43" ist , dann "die Zustandsvariable des Roboters" muss unmittelbar "auf 0 gesetzt" werden .*

```
AG ( (rdy_in & Robot_state=43) -> A[!rdy_plc U (rdy_plc &
Robot_state=0) ] )
```

*Informelle Spezifikation weiterer Eigenschaften:*

Sicherheitskritische Punkte

- Einschränkung der Bewegung der Roboterarme,
- Roboterarme müssen beim Drehen hinten sein,
- Kollision mit Presse beim Ausfahren der Arme.

*Formale Spezifikation mit dem SFS-Editor*

*/* >>>Satz 149<<< (einfaches Verbot - selbstbegrenzt) */*

*Solange "der Roboterarm 1" "hinten" ist , darf "das Einziehen des Roboterarms 1" niemals "gestartet" werden .*

```
AG !( rdy_plc & Arm1_ext=0.0 & (Arm1_backward) )
```

*/* >>>Satz 150<<< (einfaches Verbot - selbstbegrenzt) */*

*Solange "der Roboterarm 1" "in der Presse" ist , darf "das Ausfahren des Roboterarms 1" niemals "gestartet" werden .*

```
AG !( rdy_plc & Arm1_ext=0.65 & (Arm1_forward) )
```

*/* >>>Satz 151<<< (einfaches Verbot - selbstbegrenzt) */*

*Solange "der Roboterarm 2" "hinten" ist , darf "das Einziehen des Roboterarms 2" niemals "gestartet" werden .*

```
AG !( rdy_plc & Arm2_ext=0.0 & (Arm2_backward) )
```

*/* >>>Satz 152<<< (einfaches Verbot - selbstbegrenzt) */*

*Solange "der Roboterarm 2" "in der Presse" ist , darf "das Ausfahren des Roboterarms 2" niemals "gestartet" werden .*

```
AG !( rdy_plc & Arm2_ext=0.79 & (Arm2_forward) )
```

*/* >>>Satz 153<<< (erweiterte Möglichkeit - Zustand) */*

*"die Drehung des Roboters nach links" darf nur "gestartet" sein , wenn gleichzeitig "der Roboterarm 1" "hinten" ist und "der Roboterarm 2" ist "hinten" .*

```
AG !( rdy_plc & !(Arm1_ext=0.0 & Arm2_ext=0.0) &
(Robot_left) )
```

*/* >>>Satz 154<<< (erweiterte Möglichkeit - Zustand) */*

*"die Drehung des Roboters nach rechts" darf nur "gestartet" sein , wenn gleichzeitig "der Roboterarm 1" "hinten" ist und "der Roboterarm 2" ist "hinten" .*

```
AG !( rdy_plc & !(Arm1_ext=0.0 & Arm2_ext=0.0) &
(Robot_right) )
```

*/* >>>Satz 155<<< (einfaches Verbot - Zustand) */*

*Wenn "der Roboter" "mit dem Arm 1 vor der Presse" ist und "die Bereitschaftsmeldung 'Presse ist für Beladung bereit'" ist "nicht gesetzt" , dann darf "das Ausfahren des Roboterarms 1" nicht gleichzeitig "gestartet" sein .*

```
AG !( rdy_plc & Robot_angle=-90 & !Press_ready_take &
(Arm1_forward) )
```

*/* >>>Satz 156<<< (einfaches Verbot - Zustand) */*

*Wenn "der Roboter" "mit dem Arm 2 vor der Presse" ist und "die Bereitschaftsmeldung 'Presse ist zur Übergabe eines Teils bereit'" ist "nicht gesetzt" , dann darf "das Ausfahren des Roboterarms 2" nicht gleichzeitig "gestartet" sein .*

```
AG !( rdy_plc & Robot_angle=0 & !Press_ready_give &
(Arm2_forward) )
```

### *A.7.8 Geräteebene 3 - Steuerung der Presse*

Die Steuerung der Presse erhält von der übergeordneten Zellensteuerung Startsignale zur Ausführung bestimmter Aktionen. Durch verschiedene Sensoren und Aktoren ist die Steuerung der Presse mit den Maschinenelementen verbunden.

*Gegebene Daten:*

| Technische Bezeichnung | Variable | Datentyp | Kommentar |
|---|---|---|---|
| Meldung „Presse bereit zur Übernahme eines Rohteils“ | `Press_rdy_take` | Bool | |
| Meldung „Presse bereit zur Übergabe eines Fertigteils“ | `Press_rdy_give` | Bool | |
| Startsignal „Pressen“ | `Press_run_work` | Bool | |
| Startsignal „Beladen Vorbereiten“ | `Press_run_prepare` | Bool | |

*Definition neuer Daten mit dem SFS-Editor:*

Sensoren der Presse

| Technische Bezeichnung | SPS-Variable | Datentyp | Kommentar |
|---|---|---|---|
| Schalter untere Pressenposition | `Press_bottom` | Bool | TRUE - Presse unten |
| Schalter mittlere Pressenposition | `Press_middle` | Bool | TRUE - Presse in der Mitte |
| Schalter obere Pressenposition | `Press_top` | Bool | TRUE - Presse oben |

*Tabelle 37 - Datenebene 4: Sensoren der Presse*

```
%%96 -BOOL -VAR
"Press_bottom" : "die Presse"
TRUE_I : "unten"
FALSE_I : "nicht unten"

%%97 -BOOL -VAR
"Press_middle" : "die Presse"
TRUE_I : "in der Mitte"
FALSE_I : "nicht in der Mitte"

%%98 -BOOL -VAR
"Press_top" : "die Presse"
TRUE_I : "oben"
FALSE_I : "nicht oben"
```

Aktor der Presse

| Technische Bezeichnung | SPS-Variable | Datentyp | Kommentar |
|---|---|---|---|
| Motor Pressenhub (L/R) | Press_upward | Bool | TRUE - Presse schließt sich |
| | Press_down | Bool | TRUE - Presse öffnet sich |

*Tabelle 38 - Datenebene 4: Aktor der Presse*

```
%%99 -BOOL -VAR
"Press_upward" : "das Schliessen der Presse"
TRUE_O : "gestartet"
FALSE_O : "gestoppt"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"

%%100 -BOOL -VAR
"Press_down" : "das Öffnen der Presse"
TRUE_O : "gestartet"
FALSE_O : "gestoppt"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
```

*Informelle Spezifikation:*

Der Steuerung der Presse verbleibt solange in Ruhe, bis durch die Zellensteuerung entweder das Startsignal zum Presse eines Rohteils oder zur Positionierung der Presse für eine Übernahme eines neuen Rohteils erfolgt. Danach wird die geforderte Routine bearbeitet. Es sind zwei separate Routinen zu unterscheiden, die alternativ voneinander ablaufen können. Die innerhalb dieser Routinen abzuarbeitenden Teilvorgänge sind unbedingt in der festgelegten Reihenfolge abzuarbeiten, es bietet sich die Entwicklung einer Ablaufstruktur an.

*Defintion neuer Daten mit dem SFS-Editor:*

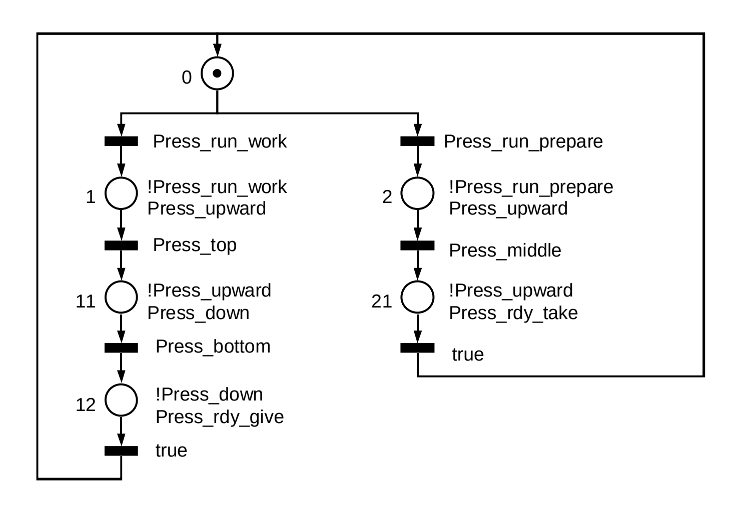
Interne Variable der Steuerung der Presse

| Technische Bezeichnung | SPS-Variable | Datentyp | Kommentar |
|---|---|---|---|
| Zustandsvariable der Presse | Press_state | Integer | |

*Tabelle 39 - Datenebene 4: Interne Variable der Steuerung der Presse*

```
%%101 -INT -VAR
"Press_state" : "die Zustandsvariable der Presse"
0_I : "0"
1_I : "1"
2_I : "2"
11_I : "11"
12_I : "12"
```

```
21_I : "21"
0_O : "auf 0 gesetzt"
1_O : "auf 1 gesetzt"
2_O : "auf 2 gesetzt"
11_O : "auf 11 gesetzt"
12_O : "auf 12 gesetzt"
21_O : "auf 21 gesetzt"
```




*Formale Spezifikation mit dem SFS-Editor:*

Ablauf Presse

*/* >>>Satz 164<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable der Presse" "0" ist und "das Startsignal 'Presse soll pressen'" ist "gesetzt" , dann muss "das Startsignal 'Presse soll pressen'" unmittelbar "zurückgesetzt" werden und "die Zustandsvariable der Presse" muss unmittelbar "auf 1 gesetzt" werden und "das Schliessen der Presse" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Press_state=0 & Press_run_work) ->
A[!rdy_plc U (rdy_plc & !Press_run_work & Press_state=1 &
Press_upward) ] )
```

*/* >>>Satz 165<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable der Presse" "0" ist und "das Startsignal 'Presse soll Beladung vorbereiten'" ist "gesetzt" , dann muss "das Startsignal 'Presse soll Beladung vorbereiten'" unmittelbar "zurückgesetzt" werden und "die Zustandsvariable der Presse" muss unmittelbar "auf 2 gesetzt" werden und "das Schliessen der Presse" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Press_state=0 & Press_run_prepare) ->
A[!rdy_plc U (rdy_plc & !Press_run_prepare & Press_state=2
```

```
& Press_upward) ] )
```

*/* >>>Satz 166<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable der Presse" "1" ist und "die Presse" ist "oben" , dann muss "das Schliessen der Presse" unmittelbar "gestoppt" werden und "die Zustandsvariable der Presse" muss unmittelbar "auf 11 gesetzt" werden und "das Öffnen der Presse" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Press_state=1 & Press_top) -> A[!rdy_plc U (rdy_plc & !Press_upward & Press_state=11 & Press_down) ] )
```

*/* >>>Satz 167<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable der Presse" "11" ist und "die Presse" ist "unten" , dann muss "das Öffnen der Presse" unmittelbar "gestoppt" werden und "die Zustandsvariable der Presse" muss unmittelbar "auf 12 gesetzt" werden und "die Bereitschaftsmeldung 'Presse ist zur Übergabe eines Teils bereit'" muss unmittelbar "gesetzt" werden .*

```
AG ( (rdy_in & Press_state=11 & Press_bottom) -> A[!rdy_plc U (rdy_plc & !Press_down & Press_state=12 & Press_ready_give) ] )
```

*/* >>>Satz 168<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable der Presse" "12" ist , dann "die Zustandsvariable der Presse" muss unmittelbar "auf 0 gesetzt" werden .*

```
AG ( (rdy_in & Press_state=12) -> A[!rdy_plc U (rdy_plc & Press_state=0) ] )
```

*/* >>>Satz 169<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable der Presse" "2" ist und "die Presse" ist "in der Mitte" , dann muss "das Schliessen der Presse" unmittelbar "gestoppt" werden und "die Zustandsvariable der Presse" muss unmittelbar "auf 21 gesetzt" werden und "die Bereitschaftsmeldung 'Presse ist für Beladung bereit'" muss unmittelbar "gesetzt" werden .*

```
AG ( (rdy_in & Press_state=2 & Press_middle) -> A[!rdy_plc U (rdy_plc & !Press_upward & Press_state=21 & Press_ready_take) ] )
```

*/* >>>Satz 170<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable der Presse" "21" ist , dann "die Zustandsvariable der Presse" muss unmittelbar "auf 0 gesetzt" werden .*

```
AG ( (rdy_in & Press_state=21) -> A[!rdy_plc U (rdy_plc &
Press_state=0) ] )
```

*Informelle Spezifikation weiterer Daten:*

Sicherheit Presse

- Bewegungsbegrenzung bei Pressenhöhe
- Kollision mit Roboterarmen muss vermieden werden

*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 171<<< (einfaches Verbot - Zustand) */*

*Wenn "die Presse" "unten" ist , dann darf "das Öffnen der Presse" nicht gleichzeitig "gestartet" sein .*

```
AG !( rdy_plc & Press_bottom & (Press_down) )
```

*/* >>>Satz 172<<< (einfaches Verbot - Zustand) */*

*Wenn "die Presse" "oben" ist , dann darf "das Schliessen der Presse" nicht gleichzeitig "gestartet" sein .*

```
AG !( rdy_plc & Press_top & (Press_upward) )
```

*/* >>>Satz 173<<< (erweiterte Möglichkeit - Zustand) */*

*"das Schliessen der Presse" darf nur "gestartet" sein , wenn gleichzeitig "der Roboterarm 1" "hinten" ist und "der Roboterarm 2" ist "hinten" .*

```
AG !( rdy_plc & !(Arm1_ext=0.0 & Arm2_ext=0.0) &
(Press_upward) )
```

### *A.7.9 Geräteebene 3 - Steuerung des Ablagebands*

Die Steuerung des Ablagebands erhält von der übergeordneten Zellensteuerung Startsignale zur Ausführung bestimmter Aktionen. Durch verschiedene Sensoren und Aktoren ist die Steuerung des Ablagebands mit den Maschinenelementen verbunden.

*Gegebene Daten:*

| Technische Bezeichnung | Variable | Datentyp | Kommentar |
|---|---|---|---|
| Meldung „Ablageband fertig zur Übernahme eines Teils“ | `Dbelt_rdy_take` | Bool | |
| Startsignal „Ablageband transportiert Teil zum Kran“ | `Dbelt_run` | Bool | |
| Meldung „Ablageband für Übergabe eines Fertigteils bereit“ | `Dbelt_rdy_give` | Bool | |

*Definition neuer Daten mit dem SFS-Editor:*

Sensor des Ablagebands

| Technische Bezeichnung | SPS-Variable | Datentyp | Kommentar |
|---|---|---|---|
| Lichtschranke am Ablageband | `LB_at_dbelt` | Bool | TRUE - Lichtschranke frei |

*Tabelle 40 - Datenebene 4: Sensor des Ablagebands*

```
%%102 -BOOL -VAR
"LB_at_dbelt" : "die Lichtschranke am Ende des Ablagebands"
TRUE_I : "frei"
FALSE_I : "blockiert"
```

Aktor des Ablagebands

| Technische Bezeichnung | SPS-Variable | Datentyp | Kommentar |
|---|---|---|---|
| Antriebsmotor Ablageband | `DBelt_start` | Bool | |

*Tabelle 41 - Datenebene 4: Aktor des Ablagebands*

```
%%103 -BOOL -VAR
"DBelt_start" : "der Antrieb des Ablagebands"
TRUE_I : "aktiv"
FALSE_I : "nicht aktiv"
TRUE_O : "gestartet"
FALSE_O : "gestoppt"
```

*Informelle Spezifikation:*

Die Steuerung des Ablagebands verbleibt solange in Ruhe, bis durch die Zellensteuerung entweder das Startsignal zum Transport eines Fertigteils zum Bandende erfolgt. Danach wird die geforderte Transportroutine bearbeitet. Die innerhalb dieser Routinen abzuarbeitenden Teilvorgänge sind unbedingt in der festgelegten Reihenfolge abzuarbeiten, es bietet sich die Entwicklung einer Ablaufstruktur an.

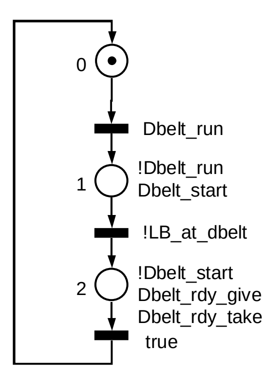
*Definition neuer Daten mit dem SFS-Editor:*

Interne Variable der Steuerung des Ablagebands

| Technische Bezeichnung | SPS-Variable | Datentyp | Kommentar |
|---|---|---|---|
| Zustandsvariable des Ablagebands | `Dbelt_state` | Integer | |

*Tabelle 42 - Datenebene 4: Interne Variable der Steuerung des Ablagebands*

```
%%104 -INT -VAR
"Dbelt_state" : "die Zustandsvariable des Ablagebands"
0_I : "0"
1_I : "1"
2_I : "2"
0_O : "auf 0 gesetzt"
1_O : "auf 1 gesetzt"
2_O : "auf 2 gesetzt"
```




*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 175<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Ablagebands" "0" ist und "das Startsignal 'Ablageband soll Teil transportieren'" ist "gesetzt" , dann muss "das Startsignal 'Ablageband soll Teil transportieren'" unmittelbar "zurückgesetzt" werden und "die Zustandsvariable des Ablagebands" muss unmittelbar "auf 1 gesetzt" werden und "der Antrieb des Ablagebands" muss unmittelbar "gestartet" werden .*

```
AG ( (rdy_in & Dbelt_state=0 & Dbelt_run) -> A[!rdy_plc U
(rdy_plc & !Dbelt_run & Dbelt_state=1 & DBelt_start) ] )
```

*/* >>>Satz 176<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Ablagebands" "1" ist und "die Lichtschranke am Ende des Ablagebands" ist "blockiert" , dann muss "der Antrieb des Ablagebands" unmittelbar "gestoppt" werden und "die Zustandsvariable des Ablagebands" muss unmittelbar "auf 2 gesetzt" werden und "die Bereitschaftsmeldung 'Ablageband ist zur Übergabe eines Teils bereit'" muss unmittelbar "gesetzt" werden und "die Bereitschaftsmeldung 'Ablageband ist zur Übernahme eines Teils bereit'" muss unmittelbar "gesetzt" werden .*

```
AG ( (rdy_in & Dbelt_state=1 & !LB_at_dbelt) -> A[!rdy_plc
U (rdy_plc & !DBelt_start & Dbelt_state=2 &
Dbelt_ready_give & Dbelt_ready_take) ] )
```

*/* >>>Satz 177<<< (einfache Forderung - direkt) */*

*Wenn "die Zustandsvariable des Ablagebands" "2" ist , dann "die Zustandsvariable des Ablagebands" muss unmittelbar "auf 0 gesetzt" werden .*

```
AG ( (rdy_in & Dbelt_state=2) -> A[!rdy_plc U (rdy_plc &
Dbelt_state=0) ] )
```

*Informelle Spezifikation weiterer Anforderungen:*

Sicherheit Ablageband

- Bewegungsbeschränkung am Bandende

*Formale Spezifikation mit dem SFS-Editor:*

*/* >>>Satz 178<<< (einfaches Verbot - Zustand) */*

*Wenn "die Lichtschranke am Ende des Ablagebands" "blockiert" ist , dann darf "der Antrieb des Ablagebands" niemals gleichzeitig "gestartet" sein .*

```
AG !( rdy_plc & !LB_at_dbelt & (DBelt_start) )
```